TEST
Automation & Controls
NATCOGROUP

SCADAWARE ®

MODBUS / OPEN ARCHITECTURE

REFERENCE MANUAL

Document 1220-05

Revised Jan 22, 2001

Contact Arthur Zatarain, PE
via www.artzat.com
For information on this document

TEST Automation & Controls

SCADAWARE® DOCUMENT 1220-05
MODBUS / OPEN ARCHITECTURE MANUAL

CONTENTS

INTRODUCTION

This document provides detailed information on TEST's implementation of the MODBUS RTU Communications Protocol. Modbus allows TEST's SCADAWARE® program to communicate with another manufacturer's device in an OPEN ARCHITECTURE system.

This document focuses on the SCADAWARE MODBUS interface and how it is implemented on a SCADAWARE® based RTU or computer. Please refer to companion document TEST SCADA PROTOCOL TUTORIAL for more information on TEST's proprietary protocol (TSP).

OPEN ARCHITECTURE

The term Open Architecture often appears in SCADA system documents and specifications to describe a generic method of interfacing diverse systems. Depending on the context of the statement, the term can have slightly different meanings and consequences. But the general intent is to describe a SCADA system designed to easily communicate with other types of systems, especially those of different vendors. Unfortunately, this concept is much easier to specify than it is to implement. Contrary to what is often stated, there are no clearly defined communication standards that allow instant connection of otherwise incompatible systems. Likewise, the theoretical interfaces often cited in specifications fail to recognize that the so called protocol "standards" are often more like "suggestions."

So the seemingly simple term *Open Architecture* does not refer to a standard option available on any piece of SCADA equipment. It more correctly refers to an overall system design concept where various parts of the system are allowed to do what they do best, while maintaining a minimum level of communication between components that is based on some common theme. Very often that common theme is based on the MODBUS protocol, a communications system developed by Modicon to interconnect their PLC equipment.

DIVIDING SYSTEM FUNCTIONS

A good question regarding open architecture designs is which functions are best performed by which pieces of equipment. Why should several different protocols be used in one system? What are the advantages of mixing and matching RTU, PLC, and Computer devices? These are good questions, and the answers will depend on the needs of each installation. However, there are several common factors that can be considered on almost any system.

A typical large scale SCADA system can be divided into three distinct areas. DCS systems have several more subdivisions, but these can be ignored for purposes of this discussion. The three main areas for SCADA are:

- Remote Terminal Units (RTU)
- SCADA Concentrator (Master Terminal Unit, or HOST computer)
- Management Information System (MIS)

The RTU is the hardware and associated end devices located at the field location to be monitored or controlled. A single system may have varied requirements for the RTUs, but they will usually fall into a range that can be met by two or three similar RTU devices.

The SCADA Concentrator, Master Terminal Unit (MTU), or Host computer is responsible for coordinating the activities of the RTUs in a real-time environment. A properly designed Host will handle all communications, real-time alarm display and logging, and diagnostic functions directly related to the monitoring and control functions.

The Management Information System (MIS) provides historical data storage and possibly network data distribution. This may include sophisticated operator interface software that offers enhanced graphical display and data base functions.

The needs of each portion of the overall system can best be met with different types of equipment and software implementations. Trying to make them all conform to a single low level protocol will likely cripple the benefits that made the equipment a good choice in the first place. A more sensible approach is to let each portion be served by the best equipment and then connect them as required with an Open Architecture protocol such as Modbus.

SCADA PROTOCOL BASICS

TEST's SCADA system, like any other SCADA or telemetry system, uses a precise method of exchanging commands and data. These methods are collectively called a PROTOCOL, and there are many different ones in existence. Some have evolved from the early days of telemetry, while others have been developed in the Programmable Logic Controller (PLC) industry.

TEST has developed its own protocol, TSP, that is designed to best handle the routine functions of typical industrial and oilfield SCADA applications. Admittedly, this protocol is quite a bit different from the protocols used by other developers. But these differences are what makes TEST's systems so practical and easy to use.

All protocols do basically the same thing: they allow digital information to be reliably sent between two or more devices. The exact way that each protocol performs this function is as varied as the equipment on which they are implemented. Generally speaking, each protocol evolved around a particular type of equipment or a specific application. Therefore, each protocol has advantages in certain applications but may not be as appropriate under different circumstances.

TEST SCADA PROTOCOL (TSP)

TEST SCADA PROTOCOL (TSP) is TEST's answer to the problems of typical oilfield and industrial SCADA. TSP was developed to serve a high technology Personal Computer based SCADA system first implemented in 1987. It has since evolved into a complex language of its own that is ideally suited to PC based SCADA solutions. TSP has built-in support for the many SCADA features designed into TEST systems. It also supports any type of communication media including hard wire, microwave, cell phone, radio, and fiber optics.

The SCADAWARE System Design Concepts (document 1010-03) should be consulted for specific information on TSP. This manual details the features that make a TSP system faster, more flexible, and more reliable than any comparable protocol in a similar situation. TEST is obviously very proud of its systems and the success they have achieved all over the world. TSP has only one minor flaw: it's not directly compatible with other protocols. But it can also be said that they are not compatible with TSP.

MODBUS PROTOCOL BASICS

Although there is no official standard with which to be compatible, one popular protocol has emerged as a defacto standard for connection of diverse systems. Like many other vendors, TEST has chosen the Modicon MODBUS PLC protocol as an Open Architecture interface. TEST's comprehensive implementation that allows outside systems to make best use of TEST's SCADA capabilities.

MODBUS is a very simple, low level protocol originally designed to interface closely coupled PLC equipment on hard wired communications links. The mechanism does nothing more than allow a single master unit to send and receive binary data with multiple slave units, all of which are on a single link. Although this is a primitive system rooted in 1960's technology, it is well understood and available on a wide variety of equipment. Its emergence as an Open Architecture standard attests to the simplicity and longevity of MODBUS rather than its actual capability. In short, MODBUS forms a lowest common denominator that can be shared with all systems on a single network.

MODBUS, as defined in the Modicon Document PI-MBUS-300 Rev B, Jan 1985, provides the basic functions to send and receive bit and word data over a comm link. MODBUS is not involved with the intent or content of the data

that is transported. In that sense, it is like a delivery service, where the rules of package sizes, addressing, and delivery verification are simply and clearly defined. The purpose of the data exchanges, and the manner in which they are conducted, are not part of the Modbus "Specification." SCADA systems which use the Modbus protocol as the transport mechanism must provide their own coordination and regulation of the data contents.

Data passed within Modbus packets is assumed to be well coordinated among the various users of the messages. Unlike TEST's TSP, data is not transferred with any meaningful identification other than the unit which transmitted the data. The identity of the data (i.e. which data base element is tied to the data) is not contained in a data response. This limitation limits Modbus to a Master-Slave arrangement in which one unit (the Master) requests all data transfers from all other units, which act as slaves. The Modbus master can only request data limited to streams of bits and integer words, stored in what Modbus considers a PLC data point.

MODBUS RTU AND ASCII PROTOCOL VARIANTS

There are two flavors of Modbus: RTU, which is a straight binary format, and ASCII, which is a hex character text format. The default mode for SCADAWARE is Binary RTU, which is the most common variant of the Modbus protocol. This low-level protocol sends all data in a single burst, with simple timeouts used for packet frame sequencing. Because all data is pure binary, it is possible (and likely) that certain binary combinations will appear which can adversely affect some communications systems. Any modem or other device which is sensitive to special codes will not work in a Modbus RTU (Binary) setup.

Another problem with the Modbus RTU mode is that very tight timing restrictions are placed on data blocks. Unlike almost any other communications protocol, there are no special start or end characters for a Modbus RTU message packet. Once a block starts on the communications line, it must flow continuously without interruption until complete. This may be a problem for some systems, such as DCS or multi-tasking PC's, in which the processor is not dedicated to the simple task of sending bytes down a communications line. Any break in the sequence may be detected as an end-of-message by the receiver, causing a failed transmission.

An alternative mode, Modbus ASCII, works around these problems by taking each binary byte and sending it as two Hex ASCII characters. This prevents any special characters from disrupting communications devices because the only codes used are 0 through 9, A through F, the colon (:), Carriage Return, and Line Feed. These are all simple text characters, and will work with any communications device.

ASCII mode also uses a unique start character, the colon, for each line. This is a much cleaner method of framing packets, because precise timing during the transmission is not necessary. The receiver looks for the start character, and takes in all characters until the end of line is marked with a carriage return. The exact timing of each individual character is no longer important.

The tradeoff in ASCII mode is that 2 bytes (characters) must be transmitted for each Modbus data byte, thereby doubling the transmission time. This is not normally a problem in SCADA systems which only do periodic updates, and a few milliseconds are of little consequence.

Each data map (explained later) in SCADAWARE has a setting which controls RTU or ASCII mode. All RTU's defined by that map must use the same mode. If both modes must be used on a single system, they must have their definitions stored in separate SCADAWARE Modbus maps. The choice of Protocol Mode is selected for an entire data map with the PROTOCOL ASCII or PROTOCOL RTU statement in the MAP file.

NOTE: Modbus ASCII normally requires a 7 bit, odd or even parity link. This must be specified in the SCADAWARE DAT file for the system which uses ASCII mode.

MODBUS DATA & REGISTER TYPES

The Modbus protocol handles several redundant data types, all of which are structured as bits, streams of bits, words, or streams of words. There are no other useful data types, such as floating point numbers or text strings. Systems which use Modbus to transport their data must fit their internal data types into either bits (send in 8 bit byte chunks), or words (sent in 16 bit, 2 byte chunks). Any other data type, including text data and floating point numbers, must be transported using some non-standard variant of Modbus. This limitation produces much of the confusion over using Modbus in typical SCADA systems which routinely send floating point and text data from unit to unit.

Legitimate Modbus data types are as follows:

REG TYPE	Typical Modbus Registers	Assumed Modbus Data Format
Coils	RTU: 0000-0999 PLC: 00000-09999	Single bit, off-on, assumed to be outputs from a PLC
Inputs	RTU: 1000-2999 PLC: 10000-29999	Single bit, off-on, assumed to be inputs into a PLC
Input Registers	RTU: 3000-3999 PLC: 30000-39999	Single 16 bit word containing 12 bit raw analog data, assumed to be a storage location on an analog input card.
Holding Registers	RTU: 4000-4999 PLC: 40000-49999	Single 16 bit word containing 16 bit signed or unsigned data, used within the PLC as a general purpose storage location. May also contain floating point data with 4 IEEE float bytes in 2 adjacent Holding Registers.
Daniels	7000-7999	Special 4 byte IEEE Floats (4 bytes per register).

Although all are essentially stored and transported as 8 bit data bytes, Modbus uses different commands to send and receive data associated with each type. Some confusion exists because of some data types have multiple commands available, used to send either a single register, or multiple registers. Most systems which use Modbus ignore the single register commands and use only the multiple equivalents, which can handle a single register if required. The only single register functions supported by SCADAWARE are the write 1 coil (W1C) command, in Modbus Master mode only, and the read 1 status input (RI) command. All other commands are multi-register.

MODBUS ADDRESSING

The addressing of data registers within a Modicon PLC is unique – each storage location has its own number. The PLC operates as if registers 10, 105, 4003, and 3143 were all separate and unique locations in the PLC memory table. No duplications are permitted, as this would confuse the PLC programming. Unfortunately, the Modbus protocol does not use these addresses directly. Instead, the Modbus protocol uses offsets (starting at 0) into tables set aside within the PLC for each separate data type. SCADAWARE has a built-in offset setting which simplifies addressing in systems which appear to be "off by one."

The Modbus protocol separates each of the above defined data types into separate tables, and accesses each table with a separate Modbus command number. While accessing each table, the protocol requires an offset value to determine where the data is located. Therefore, the combination of command number and offset form a unique location on Modbus. Accessing PLC register 3005 with Modbus is not done with a reference to address 3005. A specific command number will be used to specify the type of data at address 3005, coupled with an offset value of 4, indicating that register 3005 is located 4 spaces from the start of an imaginary table which begins with address 3001.

Note that the PLC addressing is "off by 1," in that the offset of register 3005 is 4, not 5, in a PLC register table that starts with 3001, not 3000. This "off by 1" problem is inconsistent among units which emulate the Modbus protocol. When aligning data in an emulation device, the value of the offset may or may not be off by 1. This will depend on whether the device is mimicking a Modicon PLC, whose tables start at 1, or an imaginary Modbus device, whose tables start at 0. Some trial and a lot of error may be required to determine the method used in any combination of hardware and software emulating the Modbus protocol. See the OFFSET command for more information on automatically adjusting addresses to compensate for the "off by one" problem.

MODBUS COMMANDS AND RESPONSES

Modbus commands can only be sent by the Master unit. A slave cannot issue a command, which precludes any "report by exception" processes unless some special version of Modbus is configured to work around this limitation. When a master sends out a command, it must provide all the information necessary for all slaves on the line to analyze the command. In most cases, only a single unit will process the command and issue a response. The only exception is a broadcast command (sent to Modbus ID 0) which is processed by all units, but responded to by none.

Each Modbus command from the master must provide all the information to uniquely specify the data sent or requested. The command conceptually consists of the following pieces:

Unit ID	The Modbus address (1-247, or 0 for broadcast)
Command	Single byte code specifying the command function
Address	Single address (or a start-end pair of addresses) locating the data on the 0-based offset table for each different type of Modbus data. Normally, Register 2 is at offset 1.
Data	Bits or Words (always sent in byte format) of information sent from master to slave

Expressed in plain talk, a Modbus command will speak to a certain unit number, issuing a specific command that affects either a single address, or a range of addresses. Commands that send data from master to slave will also contain the appropriate number of data chunks. Responses sent by the slave to the master have a similar structure, but contain only the information requested by the Master. The addresses of the returned data are, sadly, not included in the response.

Modus assumes that only a single master is doing all the talking, so therefore any response received will come from the most recently accessed slave, using the most recently requested address range. The slave answers with only the data, and not the address, assuming that the master was expecting a specific piece of information. This lack of addressing information means that other units listening to the reply will not know the associated address of the data, and cannot easily tap into data transfers over a Modbus link.

As a simple example, pretend that a Modbus exchange is done with people speaking as a group, with one specific person leading the conversation. Only one person (the master) is permitted to ask questions, and only the person addressed by the master (the slave) is permitted to reply. Assume that each person has several sheets of colored paper with numbers listed on them.

In our example, the master asks "Hey Sam, look on your blue paper and tell me the three numbers that start at position 4 on the list." Although everyone hears the question, only Sam will reply. He looks at his blue paper, counts down to the fourth position, and answers back "This is Sam, the 3 answers from the blue paper are 305, 112, and 782." Everyone hears Sam's answer, but unless they were paying close attention, they will not know what he is talking about. This is because Sam's answer contains the clue that the numbers are from the blue paper, but it does not say which positions on the list he is reporting. Note that the meaning of the numbers is also not included. The purpose of the data (i.e. value, high setpoint, deadband, etc.) is not part of the Modbus specification. The information in the data packet is arbitrary, representing prices, temperatures, or random bit patterns.

SCADAWARE supports a variety of Modbus commands which permit connection to any reasonable Modbus device or software system. The following table lists the commands supported in SCADAWARE's Mosbus Master and Slave mode:

Number	Mnemonic	Function
1	RC	Read Coil Group
2	RI	Read Single Status Input
3	RHR	Read Holding Register Group
4	RIR	Read Input Register Group
5	WIC	Write 1 Coil (master mode only)
7	RES	Read Exception Status
15	WC	Write Coil Group
16	WHR	Write Holding Register Group

The following Modbus commands are not commonly used and are not supported by SCADAWARE. They are listed here for reference only.

Number	Mnemonic	Function
6	PSR	Write Single Register
17	RID	Report Slave ID Type
19	RCL	Reset Comm Link
7	RES	Read Exception Status

COMMON MODBUS EMULATION PROBLEMS

Most products which claim to provide Modbus emulation do not support the full specification as defined in the Modicon document (PI-MBUS-300, Rev B, January 1985, available from Gould). This is understandable because much of the specification is related to PLC specific issues, such as programming and status reporting. The following are common limitations which may affect how a device is used in a SCADAWARE based Modbus system:

1 **Single Bit/Word Commands**: The Modbus specification provides many commands which are not actually required to transport data. For example, there are commands to send and receive single registers, and very similar commands to send and receive multiple registers, including only a single one. Some implementations delete the single register commands and support only the ones which permit transfer of a variable number of registers (including only 1).

2 **Multiple Bit/Word Commands**: Some systems (such as FlowAutomation devices) provide the multiple transfer commands, but do not permit more than one register per transfer. This limitation may be due to floating point number modifications in which 2 adjacent words are used to store a single 4 byte float number. If this is the case, then multiple Modbus commands may be required to transfer the necessary data.

3 **Raw Analog Data Size**: A common assumption is that Modbus Input registers represent 12 bits of unsigned integer data, in which full scale ranges from 0-4095. This is not always the case, and units will send raw data ranging from 8 to 16 bits. The effect on the receiving device will depend on how raw data is interpreted by the receiving software. SCADAWARE assumes 12 bits of raw data when processing Input Registers, and 15 data bits for Holding registers (which sometimes also have a 16th sign bit). Mapping options are provided to override these values on an individual basis. See the detailed discussion on Analog / Integer formats later in this document.

4 **Address Offsets**: Modicon PLC's use an "off by 1" addressing scheme. Register 101 in the PLC is addressed as location 100 in Modbus because location 101 is offset 100 spaces from the start of the internal data table. When mapping internal DATA onto the Modbus, some devices use the "off by 1" scheme, while others do not. Sometimes trial and error testing is required to understand exactly how a vendor relates internal data to Modbus data. Unless modified with the OFFSET command, SCADAWARE will use address 101 as Modbus address 101, with no offset, with no regard for how a PLC may position that data within its own internal table. If an application requires that Modbus address 101 refer to register 102, then the OFFSET ON command can be used in the M3R file which defines the Modbus data map.

5 **Host and Slave Mode:** Most units support either Modbus Slave mode, permitting them to respond to Modbus data requests. Fewer devices support Master mode, which is required if a device will retrieve data from another unit. SCADAWARE dynamically supports both modes in a manner very similar to how TSP sends and receives data. The software is never locked into either mode. When not requesting data, SCADAWARE waits in Slave mode. Master mode is automatically set whenever SCADAWARE requests data using a Modbus MAP SCAN or MAP DATA command.

6 **Floating Point Numbers**: Most modern computing systems use floating point numbers based on the IEEE 4 byte standard. Unfortunately, Modbus does not provide a data type of 4 bytes, nor does it support the concept of floating point numbers. Methods of dealing with floating point numbers with Modbus are covered elsewhere in this document. All of them are special cases which cannot be processed by what is considered "standard" Modbus.

7 **RTU Mode Timing**: The Modbus RTU (Binary) mode protocol relies on character pacing to establish the start and end of data blocks. Timing is also used to detect problems (lost characters) on the comm line. This awkward system demands very precise timing which is specified in terms of "character times" by the Modbus specification. Character times are determined by the baud rate, and equate to the time period required to send one 8 bit data byte. According to the Modbus spec, the RTU mode of data transfer times out after 2 character times. In practice, most systems permit less stringent communications breaks, although some time limit must be established in order for messages to be properly framed without use of start and stop characters. Proper coordination of the timing is required among all units on a link. If timing is a problem, then the Modbus ASCII mode should be considered in place of RTU Binary.

DATA MAPPING

MODBUS INTERFACE MAPPING

MODBUS defines only two low-level data types: bits and words. TSP provides high-level data types such as Analog Values and AGA-3 gas flow meter calculations. SCADAWARE also provides data point attributes which provide status information such as alarm condition, or time information, such as alarm timestamps. The interface provided by SCADAWARE allows all of the important TSP data types and attributes to be easily "mapped" into equivalent low-level Modbus coils, inputs, and registers for transport to and from external systems. Special provisions have been made to accommodate floating point data using a variety of "popular" techniques.

Using "data mapping" techniques available in SCADAWARE, the built-in TSP features such as alarm status, deadbands, setpoints, and other internal system variables are transformed into Modbus compatible values. This mapping provides a fixed relationship between the actual data values in SCADAWARE and equivalent values in a virtual Modbus data table. Outside accesses to the SCADAWARE system refer to Modbus addresses as if they actually existed within a Modicon PLC or RTU device.

MAP STRUCTURE

Each map consists of a memory table in SCADAWARE memory. Each map has certain parameters, such as tag name and Modbus RTU/ASCII settings, that are common to all Modbus ID's defined within that map. Within the map, there are up to 240 separate ID definitions. Each definition has individual parameters for the number of inputs, holding registers, etc, and where these data values link into the TSP data base. Each SCADAWARE system supports up to 10 simultaneous maps, providing room for 2400 separate Modbus units, each of which can map numerous ranges of data elements.

Modbus data accesses occur at numeric addresses which reflect locations in the virtual Modbus data tables. These addresses begin at 0 and run up to a limit determined by each separate table defined within the SCADAWARE map for each Modbus ID. Data transfers use these tables as guides in building Modbus commands, and for responding to data requests. A Modbus protocol data transfer would specify a transfer beginning at a certain addresse (i.e. table offset) and running for some number of points. The actual meaning of the data points is unknown to Modbus, but will make sense to SCADAWARE because of steering information contained in the Map tables.

The Data Mapping process allows assignment of TSP values and channel status conditions to these imaginary Modbus points. For example, we may assign the off/on value of TSP Status Points 1-8 to Modbus Inputs 0-7. We can then assign the alarm condition of those same TSP status points to the next 8 Modbus points, 8-15. With this scheme, we can pick and choose what TSP data will be accessible on the Modbus. It is even possible to have a single piece of TSP data appear more than once in the same data map. This may be convenient in complex systems that scan some data often and other data only occasionally. Regardless, the mapping configuration is completely up to the system designer.

MAP CONFIGURATION

The system designer must decide how the SCADAWARE data will be mapped into the equivalent MODBUS data types. Although the Modbus choices are limited to bits and words, SCADAWARE's Modbus system allows a convenient and flexible way to let internal TSP data appear as normal Modbus data. This process is called mapping. TSP data is mapped into suitable Modbus data locations such that it appears to reside in a normal Modicon PLC or RTU device.

MAP Configuration is done with a simple ASCII text file similar to other setup files in the TEST SCADA environment. The file contains all of the setup and TSP-to-MODBUS assignments in a structure referred to as a Data Map. The system can support several simultaneous data maps, although only one will be required for most systems. The data map provides logical linkages between TSP information and Modbus addresses. With the map system, an outside system can access Modbus data without concern for the actual location within the SCADAWARE internal structure.

Data Mapping requires some planning to insure that the proper TSP data is available in a form convenient to the external system. The logical layout of channel values, setpoints, and other TSP data is completely up to the designer. The mapping system is very flexible, but requires knowledge of the external system's needs for the TSP data.

The configuration file can have any name, although one with the filetype of "MBR" (for ModBus RTU) is recommended. For example, a system named "HOST" should have a file called "HOST.MBR" in which all mapping information is contained. This file is processed by a MAP LOAD command executed by any RTU type task on the system. This will normally be done by Task 0 during startup, but this is not absolutely necessary. Loading of a map file is

a one-time requirement, and it can be done only once per session. *If changes are made to a map file, the system must be restarted so that the map tables can be rebuilt during system startup.*

Maps are global to the entire SCADAWARE program. Once a MAP is loaded, it can be referenced by any task needing the MAP information. A task acting as a MODBUS RTU will receive commands from an external unit and react to that command with the Modbus data map. This will be done automatically by the task once it selects the proper data map. Therefore, most of the work required to use the Modbus interface lies in generation of the map file.

The map file consists of text lines, each of which starts with a special keyword. The keyword tells the processor the line's function. The remainder of the line provides additional information that relates to that keyword. The MAP LOAD process will read the entire file and build the data map based on the information in the file. The interpretation of the file can be verified with the MAP DUMP command which produces an ASCII text file which can be viewed or printed for framing on the wall.

MODBUS FLOATING POINT VARIATIONS

The "official" Modbus specification does not provide support for floating point numbers. This limitation is overcome by various vendors in slightly different ways. All variations supported by SCADAWARE use 4 byte IEEE floating point numbers. The 4 bytes are transferred over Modbus in a variety of methods, none of which are part of the basic Modbus protocol definition. The variations either use two adjacent register locations to hold the 4 byte float, or they use special 4 byte registers in place of the standard 2 byte registers. SCADAWARE uses a protocol keyword option to specify which variation applies to all RTU's in that data map.

The variations supported by SCADAWARE are:

Default:	2 adjacent Holding or Input registers hold one 4 byte float
Swapped:	2 adjacent Holding or input registers, swapped from the Default mode
Daniels:	4 byte registers at artificial addresses 7000-7999
FlowAutomation	4 byte registers at any Holding Register address

The format of the IEEE float number can be divided into 4 separate bytes, noted as A-D as shown below:

```
    A         B         C         D
Seeeeeee  eeeeeeee  fffffff   fffffff
```

where s = Sign, e = exponent, and f = fractional part of the IEEE number. All protocols except Swapped assume the 4 bytes appear on the Modbus line in ABCD order. Swapped assumes they are in CDAB order.

DEFAULT FLOATING POINT FORMAT

One popular method of handling floating point numbers on Modbus is to store the 4 byte IEEE float in 2 adjacent registers in the Holding Register table. This method of floating point transfer is the default for SCADAWARE, and can be selected with the PROTOCOL INTEGER or PROTOCOL NONE statement in the Map definition file. The internal mechanisms used by the sender and receiver must be in sync in order for the floating point numbers to be transferred. In these exchanges, Modbus is acting simply as a data carrier. The actual contents of the registers make sense only to the programs exchanging the data.

The alignment of these values can be very complex, especially when programs from two different manufacturers are exchanging floating point data over Modbus. Fortunately, SCADAWARE takes care of the complex addressing required to stuff one float value into 2 integer registers. The mapping is done with TSP data references, and starting points in the Modbus table. All other calculations necessary to position the floating point numbers is handled by the SCADAWARE Modbus driver.

DANIELS VARIATION

Instead of stuffing one float into two registers, Daniels defines a whole new set of 4 byte Modbus registers starting at address 7000. Their system allows mapping of internal flow computer values to this new class of Modbus registers. When the Daniels Protocol is specified to SCADAWARE, the driver uses this special addressing method to set and retrieve these special "7000 class" floating point registers. These transfers are not actual Modbus messages, but use a similar format which is accommodated by the SCADAWARE driver.

Normal Modbus transfers are also possible in a Daniels system, but they are limited to bits and words just like any ordinary Modicon device. The SCADAWARE driver will treat any transfers below address 5000 as if they were normal Modbus addresses. This allows normal Modbus and Daniels variants to be mixed in the same data table. Note that a memory saving table offset can be specified to save the space between 0 and 7000 when using the special Daniels registers. Details on this offset are provided in the section on defining Modbus data tables in the Map file.

FLOWAUTOMATION VARIATION

The FlowAutomation variation is similar to the Daniels variant, but permits floating point numbers within the normal holding register address range. Most FlowAutomation systems restrict data transfers to single values per command. The command line from the master must be formatted to request a single Holding Register. The FlowAutomation device responds with 4 bytes containing the IEEE float. Unlike Daniels, the address of these 4 byte registers can be anywhere in the Holding Register table.

When FlowAutomation mode is in use, normal Modbus integer format data cannot be used on the same Modbus ID because all holding and input registers will be in the 4 byte format.

MODBUS MAP FILE

MAP FILE FORMAT

The SCADAWARE map file is a text configuration file which tells the driver how to relate the TSP addresses to Modbus addresses on one or more Modbus units. The file contains plain text statements on lines which begin with Map file keywords. Each line performs a specific function to build and define the tables which form the data map between TSP and Modbus. The MAP file can be processed by any task, although normal practice is to have it processed by TASK 0 during system startup.

The format of the file is very similar to other TSP text files. The keyword is the first word on the line, and some abbreviations are allowed. A semicolon (;) can be used as a comment character, allowing for comments anywhere in the text file. *Comments in Map files are highly recommended.*

MAP FILE KEYWORDS

NOTE: The MAP FILE keywords listed below are processed by SCADAWARE during execution of a TSP MAP LOAD MODBUS command. They cannot be processed by TSP directly, and only make sense in the context of the MAP file processing. Do not confuse these keywords with those associated with the TSP command MAP, which is explained elsewhere in this document. A sample MAP file is also provided for reference.

MSG Text message

Text messages can be sent to the system console during map file processing with the MSG statement. All text after the MSG keyword is echoed to the system console.

```
Msg Starting Modbus Setup for Production Facility PLC
```

DELAY Specify the Inter-Character Timeout Parameter

DELAY sets the maximum number of ticks (50 ms periods) allowed between incoming characters during processing of an incoming Modbus RTU line. Because the RTU protocol mode has no start and end character, timing alone determines the boundary of a command line. The default delay is 2 system ticks, equivalent to about 100 milliseconds. An internal timer is started when a unit receives the first character of an incoming message. This timer is reset each time another character is received. If the set number of system ticks has elapsed between incoming characters, the input buffer will be cleared and the current command canceled. The system will assume a problem has occurred and begin waiting for the next command.

```
Delay 8    ; set delay to 8 * 50 ms for incoming characters.
```

ID Specify Modbus RTU Address

Specify the ID number to be used for subsequent map building statements in the map file. No table sizes or data maps can be defined until an ID has been specified. Each subsequent ID command terminates generation of the current map and starts building the next ID setup.

```
ID 4      ; start defining Modbus Address 4
```

OFFSET Controls Automatic Address Offset to fix "off by one"

In most applications, the Modbus addresses used in the Map file are the actual addresses which will be sent out over the data link during Modbus protocol transactions. Some systems behave like a PLC rather than an RTU in that they use an offset value rather than a true address when referencing internal registers. On those systems, register 12 is accessed at Modbus offset 11, because address 12 is offset by 11 spaces from the table. If no automatic offset is used, the programmer must keep track of the register address as it differs (by 1) from the Modbus address. To simplify programming on PLC type systems, SCADAWARE provides an automatic offset adjustment to addresses as they appear in Modbus command lines. When OFFSET is turned on, the programmer can set up the map file and download files using the address values inside the PLC rather than a value which is 1 less. With OFFSET turned on, the programmer can reference PLC register 100, and SCADAWARE will send out 99 on the link. An incoming address of 51 will be dealt with inside SCADAWARE as if register 52 were the one being used.

```
ID 4    ; start unit number 4
Offset ON ; this unit accesses register 10 as modbus address 9
| | | | | | | |
ID 11
Offset off    ; this unit accesses register 6 as modbus register 6
```

PROTOCOL Modbus Variations

Specify a variation from the default protocol variation of Modbus RTU. Current choices are:

```
RTU (or BINARY)    RTU mode for all ID's in this map
```

```
Swap                Default Float mode with 2 regs swapped
ASCII               ASCII mode for all ID's in this map
Daniels             Daniels Floating Point mode
FlowAuto            FlowAutomation Floating Point Mode
Integer (or None)   Default Float Mode using 2 Adjacent Holding
                    Registers
```

Note that the RTU and ASCII options affect the entire data map. Other Protocol options affect the currently defined ID, and all subsequent ID's unless a new protocol option is later specified. This "sticky" assignment is provided for compatibility with older installed systems in which protocol options applied to the entire map, and not just specific ID tables

SELE Select Logical TSP RTU

Select a logical TSP RTU which is used in subsequent TSP data table references. This is a convenience so that subsequent map lines do not have to explicitly name a particular SCADAWARE RTU on each line. The RTU named in the SELECT line will be the default RTU.

```
Sele WD34A     ; move to West Delta 34 A RTU for TSP references
```

TABLE Specify Modbus Map Table

Each Modbus ID within the MAP has its own tables for each of the Modbus data types. For each separate ID, the size of these tables must be declared prior to use of each data type. The TABLE statement specifies the size (and sometimes the starting address) of the Modbus map table for each data type. Required parameters include a keyword specifying the type of table being defined, and a number specifying the size of the table. An additional, optional parameter specifies a starting address offset which is added to values specified in the new table. This offset setting is designed to save memory space and processing time for tables with high addresses on which the low addresses are never used. So instead of allocating a total of 7010 spaces for a Daniels which uses only 10 spaces (7001 through 7010), the system will allocate only 10 spaces, starting at 7001.

Available table definition keywords are:

COIL	Modbus Output coil	(digital, off or on)
INPUT	Modbus Input point	(digital, off or on)
HREG	Holding Register	(16 bit word or integer)
IREG	Input Register	(16 bit word or integer)

If more than one table definition exists for the same type of table under the same map ID, only the first definition line for that table will be accepted.

```
Table Hreg 100  ; table for 100 Holding Registers starting at 0
Table Ireg 50 7001  ; table for 50 entries, start is daniels 7001
Table Coil 32       ; room for 32 output coils
Table Input 32      ; room for 32 status inputs
```

The optional table offset can be used to reduce the overall size of the table, and also to simplify address moves or duplication in similar systems. If a Modbus data table always exists in a certain sequence, but resides at different starting addresses, it can be defined in a "generic" manner starting at an arbitrary address such as 0 or 1. The actual target address can be pinpointed by specifying an OFFSET address in the Table statement line. This offset will be ADDED to whatever addresses are provided in the table setup. Consider the following setup:

```
Table IREG 10  100      ; 10 table entries, starting at 100
```

```
map ireg 1 v1:v9          ; 9 Input registers map to value channels
```

In the example, a sequence of 9 value channels are mapped into 9 Input registers in the Modbus data table. The actual addresses of these channels will be:

TSP	MODBUS
Unused	100
V1	101
V2	102
V3	103
V4	104
V5	105
V6	107
V8	108
V9	109

Note that Modbus address 100 was allocated, but not used in this example. This was done to illustrate that the quantity of addresses specified (10) will begin at the Modbus address specified in the offset (100). If no offset is provided, a default of 0 is used, causing the table to begin at Modbus address 0.

In the example, value channels 1 through 9 were mapped to the list beginning at local table address 1. However, because of the offset specified on the TABLE line, the final address (on the Modbus) for value channel 1 (V1) becomes 1 plus 100, or 101. This offset feature can reduce confusion over mapping in devices which have repetitive sequences of similar registers, such as a multi-channel flow meter. The same sequence appears in each logical meter, but with a different starting address for each meter. With the offset specification, multiple tables can be easily defined, with the only difference in each being the offset address. This makes changes and moves less complex because only the group start address must be manipulated, rather than the individual addresses of all the group members.

Another benefit of using table address offsets is that the table size is reduced, thereby saving memory and improving processing speed. If only addresses 7000-7010 are used, defining a table with the range of 0-7010 would waste nearly all of the memory required for the table. Specifying a table of size 11, with starting address of 7000, reduces the table memory requirement to nearly nothing.

TAG Table Reference ID

Each SCADAWARE map can have a unique name to replace the default name of MAP1, MAP2, etc, assigned during startup. The TAG statement sets the tag name for the data map being defined. These tag names can be used by any RTU type task to select a data map (using MAP SELE XXXX) by tag name rather than by index number.

```
Tag  PLC1      ; Tag for this table is PLC1
```

UNITS Define number of Modbus ID's in the overall table

Within each data map, up to 240 separate Modbus IDs can be defined, each with its own map setup containing many separate data linkages. The UNITS command defines how many logical Modbus station IDs will be emulated by the data map being defined. If this command is omitted, a value of 4 is assumed. This statement should appear at the top of the MAP file, and must appear before any ID's are defined.

```
Units 5      ; this table will emulate 5 different Modbus addresses
```

MAP Link TSP to Modbus

Specify a logical link between TSP values in a previously defined table and specific Modbus locations.

Parameters include a keyword specifying the Modbus data type, a number representing the position within that Modbus data table, a TSP channel or channel range, and optional data type specifications. Available Modbus data type keywords are:

COIL	Modbus Output coil	(digital, off or on)
INPUT	Modbus Input point	(digital, off or on)
HREG	Holding Register	(16 bit word or integer)
IREG	Input Register	(16 bit word or integer)

Following the Modbus data type is a number which represents the position within that Modbus data table, for the current Modbus ID. Remember that SCADAWARE's Modbus tables begin with address 0, so the Modbus tables start at position 0 and extend up to the maximum declared table size minus 1.

The next parameter defines the TSP channel(s) which are linked to the specified Modbus address(es). The reference can be either a single TSP channel or a channel range. If a single channel is given it is directly mapped with the specified Modbus table and position. If a channel range is specified, the first channel in the range is mapped to the specified Modbus table and position. Consecutive channels in the channel range are mapped to consecutive positions in the same Modbus table.

For example, consider the following lines which could be used to map TSP Output channels 1-5 with Modbus Coil table positions 0-4. In this example each line maps a single TSP Output channel.

```
MAP Coil 0 O1
MAP Coil 1 O2
MAP Coil 2 O3
MAP Coil 3 O4
MAP Coil 4 O5
```

By using a channel range, the same 5 TSP Output channels could be mapped to the same 5 Modbus Coil positions in a single command as shown below.

```
MAP Coil 0 O1:O5
```

The remaining parameters of a MAP specification statement are optional. These parameters are keywords which specify exactly what data pertaining to the referenced TSP channels will be mapped to the Modbus tables. These parameters can also control the format of the data mapped to Modbus Input registers and Holding registers. These options are discussed below.

BIT DATA MAPPING OPTIONS

For Modbus **Coils** and **Inputs**, each location contains either a 0 or a 1. Therefore, each location can be mapped with the value of a digital TSP channel (Status Input or Output) or the status of any channel's condition, such as the existence of a particular alarm condition. The available keywords for specifying which channel data will be mapped into each Modbus Coil and Input location are listed below. Following each keyword is the condition for which the corresponding Modbus locations will contain a 1 when the data is transferred from TSP to Modbus.

KEYWORD	CHANNEL CONDITION
Value	1 of current value of channel <> 0
New	1 if channel in alarm but not yet acknowledged
Alarm	1 if acknowledged but still in alarm state
Res	1 if no longer in alarm and waiting for reset
Abnormal	1 if alarm conditions NEW, ALARM, or RESET exist

If no keyword is specified, the keyword VALUE is assumed. For Coils and Inputs, multiple keywords can be specified on a single line. In such a case, the associated Modbus location would contain a 1 if any of the conditions are met at the time of the transfer from TSP to Modbus. For example, consider the following 3 map setup lines:

```
MAP Coil 0 O1 Value       ; 1 if channel is ON
MAP Coil 1 O1 New         ; 1 if channel in New Alarm condition
MAP Coil 2 O1 Value New   ; 1 if channel Abnormal or Unacknowledged
```

Consider the value of Modbus Coil locations 0, 1, and 2 following an update of these 3 locations from a TEST SCADA unit. If the value of Output channel 1 is 1, Coil 0 would contain a 1, otherwise a 0. If Output channel 1 is in alarm but has not yet been acknowledged, Coil 1 would contain a 1, otherwise a 0. If either the value of Output channel is 1, or Output channel 1 is in alarm but has not yet been acknowledged, Coil 2 would contain a 1, otherwise a 0.

An update from TSP to Modbus can occur in 1 of 2 ways. First, a request can be sent to a TEST SCADA unit acting as a slave. The TEST unit would interpret the request, build up a response in Modbus format, and send back the response. Second, the TEST SCADA unit can act as a master and simply build up a Modbus command and send it out.

Now let us consider what happens when data is transferred in the opposite direction, from Modbus to TSP. When data is transferred in this direction, only the Modbus locations that are mapped to TEST SCADA channel VALUES will have an effect on the TEST SCADA unit. For example, again consider the following 3 map setup lines.

```
MAP Coil 0 O1 Value
MAP Coil 1 O1 New
MAP Coil 2 O1 Value New
```

Consider the state of SCADAWARE after an update from these 3 Modbus locations. If the value of Coil 0 is 1, Output channel 1 1 would contain a 1, otherwise a 0. The update from Coil 1 would have no effect on the TEST SCADA unit because the mapping for that Modbus point does not contain a TSP channel value. The update from Coil 2 would have the same effect as the update from Coil 0. This is because the NEW parameter would be ignored and the VALUE parameter would work the same as it did for Coil 0.

Since Coils and Inputs contain either a 0 or a 1, they are easily mapped with the Status Input and Output channels of a TEST SCADA system. However, Coils and Inputs can also be mapped to any other type of TEST SCADA channel. Transferring data for value type channels is exactly the same as transferring data for digital type channels. When going from TSP to Modbus, a Modbus location will be set to 1 when the value of the corresponding channel is not equal to 0, otherwise it will be set to 0. When going from Modbus to TSP, a channel value will be set to 1 when the Modbus location contains a 1, otherwise it will be set to 0.

WORD DATA MAPPING OPTIONS

For Modbus **Input Registers** and **Holding Registers**, each location contains an integer value, always contained in 16 bit field (also called a "word"). What each integer represents depends on the map setup for each location. The available keywords for specifying what data will be mapped into each Input and Holding register location are listed below

KEYWORD	CHANNEL PARAMETER
Value	Integer representation of channel's current value
Hi	Channel's High alarm setpoint
Lo	Channel's Low alarm setpoint
Db	Channel's Deadband value
Float	Indicates that this channel will be sent as a Floating Point number
12-16	Overrides default word size for Input and Holding Registers. Input registers default to 12 bits. Holding registers default to 16 bits.

If no keyword is specified, the keyword VALUE is assumed. Data will assumed to be Integer unless the FLOAT option is specified. Integer format means the value will be represented by a 15 data bit integer value (perhaps in a 16 bit signed integer field) which occupies a single Modbus register. The FLOAT keyword will specify a Modbus extension in which values are represented by a 4 byte (32-bit) floating point number.

Note that Floating point data types are not part of the Modbus specification. One common method of providing floating point support is to store the 4 bytes (2 words) of the IEEE floating point value into two adjacent 16 bit Modbus registers. This 32-bit value occupies 2 consecutive 16 bit registers, where the upper 2 bytes are stored in the specified location and the lower two bytes will be stored in the next consecutive location. Unfortunately, alternate formats also exist. Some of these can be processed by SCADAWARE as explained below.

For example, consider the following map setup line which does not use floating point:

```
MAP HREG 0 V1:V8
```

This line will map the values of TSP Value channels 1-8 with Modbus Holding Registers 0-7. Each holding register will contain a 16-bit integer representing the value of each channel. The map defined by this line would look like this:

MODBUS HOLDING REGISTERS	TSP VALUE CHANNELS
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8

Now consider the following map setup line, which does use floating point data:

```
MAP HREG 0 V1:V4 FLOAT
```

This line will map the values of TSP Value channels 1-4 with Modbus Holding Registers 0-7. Two consecutive register locations would be used to represent the value of each channel in a 32-bit format. The map defined by this line would look like this:

MODBUS HOLDING REGISTERS	TSP VALUE CHANNELS
0	V1 - upper 2 bytes
1	V1 - lower 2 bytes
2	V2 - upper 2 bytes
3	V2 - lower 2 bytes
4	V3 - upper 2 bytes
5	V3 - lower 2 bytes
6	V4 - upper 2 bytes
7	V4 - lower 2 bytes

SCADAWARE will ignore attempts to access data across floating point boundaries. In the example above, attempts to read or write to holding registers 3-4 with a single value will be ignored because the map understands that these registers relate to 2 different TSP values.

When setting up a map for registers, only one of the keywords VALUE, HI, LO, and DB should be specified for a single Modbus range. The FLOAT keyword can be used by itself, before, or after any one of the other keywords. However, if two of the other keywords are specified on the same line, the second one is the one that will have precedence. The following examples are given to help illustrate this point:

FILE STATEMENT	DATA REFERENCE
MAP HREG 0 V1	chan value in integer format
MAP HREG 8 A1 FLOAT	chan value in float format
MAP HREG 6 M1 FLOAT HI	hi setpoint in float format
MAP IREG 0 V1 LO FLOAT	lo setpoint in float format
MAP IREG 2 A1 FLOAT VAL	chan value in float format
MAP IREG 4 M1 VALUE HI	hi setpoint in integer format

ANALOG / INTEGER SIZE AND FORMATS

Modbus does not fully define integer formats that are used for analog conversions and general integer use. "Standard" Modbus provides for Input Registers (analog inputs) to be 12 bits, unsigned, right justified in a field of 16 total bits. The default 12 bit Input Register relates to the normal "raw" data provided by the A/D converter in an unspecified Modicon PLC or RTU. *However, it is important to understand the exact details of the integer format being used in any application.* Consult the hardware and software manuals for each device to determine the exact bit mapping of all data formats used with SCADAWARE.

The "standard" Modbus integer is a 12 bit unsigned number. A 12 bit size suggests that the full scale swing of the Input Register is 0-4095. For the raw A/D, 0 is 0, 2048 is half scale, and 4095 is full scale. SCADAWARE assumes this 12 bits as the default format for Input Registers. Variations from this simple 12 bit format can be handled as discussed below. Regardless, all Modbus integers are converted upon receipt to meet the normal 16 bit signed integers used within the program. The reverse is done when transmitting integers from SCADAWARE.

Modbus Holding Registers are generally assumed to be 15 data bits plus a sign bit, but the integer format within those bits is undefined by the Modbus specification. SCADAWARE assumes holding registers contain 16 bit signed integers, with a full scale swing is ±32,768. This assumption means that Modbus Holding Registers are directly compatible with SCADAWARE, so no input/output conversion is required. Variations can be handled as discussed below.

It is common to find alternate integer formats in real world Modbus applications. To handle the variants, SCADAWARE provides MAP command options to specify alternate integer sizes for each Input and Holding register group. The option is entered as a number on the MAP IREG or MAP HREG statement in the map file. The default is 12

data bits for Input Registers, and 15 data bits for Holding registers. The optional number can range from 1-15. When processing integer data for Ireg and Hreg addresses, SCADAWARE will shift the integer's bits left as required to meet the needs of SCADAWARE (incoming data), or right to meet the needs of Modbus (outgoing data). Details on this are provided below.

In a 12 bit unsigned Modbus register, incoming numbers will be shifted left 3 bits to produce a 16 bit signed number (the upper bit is for the sign, and is ignored). In an 8 bit system, outgoing numbers will be right shifted 7 bits to produce an 8 bit result (stored in 16 bits, upper byte all 0) for use in a Modicon type D/A analog output. However, all Input Registers and Holding Registers occupy 16 bits (2 bytes) in the Modbus data table. Single byte registers are not defined.

ANALOG / INTEGER FORMATS

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	Bit Position
A	S	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0	SCADAWARE ±32K
B	Z	z	z	z	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0	Modbus A/D 0-4095
C	S	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	z	z	z	12 bit Shifted/Signed A/D
D	S	z	z	z	b11	b10	b9	b8	v7	b6	b5	b4	b3	b2	b1	b0	12 Bit signed A/D
E	X	x	x	x	x	x	x	x	d7	d6	d5	d4	d3	d2	d1	d0	8 bit D/A Output 0-255

b = Bit Position d=Data bit S = Sign bit x = Don't care (probably 0) z = Always 0

The table above shows a sample of possible data formats found in Modbus applications. The SCADAWARE entry at row A is the standard signed integer format used in most computers. This is a signed integer with 15 data bits (32K) in the least significant bits, and a sign bit in the most significant (left most) position. Some systems do not use a sign bit which results in 16 bit "words" which range from positive 0-64K. Unsigned 16 bit words are not supported by SCADAWARE, although they can be used as signed integers in the limited range of 0-32K.

The entry at row B shows the 12 bit unsigned format described in the basic Modbus specification. When translated to a standard integer within SCADAWARE, the 12 bit number is shifted left 3 times such that it has 3 zeros in the lower bit positions. This means the resulting SCADAWARE integer will always change by 8 (3 bits worth) for every bit change of the Modbus input register. Note that the highest bit position is unused and set to 0, producing an integer range of 0-32K. The MAP file bit size for this entry is 12, which is also the default value used for all Input Registers.

The third entry at row C is typical of PLC A/D inputs which provide plus and minus conversions. Here the 12 bits of A/D conversion are shifted left within a 16 bit signed integer field to produce an integer range of ±32K. However, because the lower 3 bits are always 0, the effect is that of a 15 bit A/D with a resolution of 8. The MAP file bit size setting for this data type would be 15, indicating that no shifting is required. No data conversion is required because all 16 bits are in place, ready to be used as a standard SCADAWARE integer. *This is the only Modbus integer format which permits negative Modbus integers to be used with SCADAWARE.* This is also the integer format used as the default for Holding Registers, allowing them to easily contain 16 bit signed integer data.

The fourth entry at row D is a 12 bit A/D similar to that in row C, except that the useful data bits have not been shifted left on the Modbus side. The unused bits are placed between the data bits and the sign bit, which would produce a ±4095 range if used with software that understood this non-standard format. However, when a bit size of 12 is specified in the SCADAWARE Map file, the resulting left shift will cause the sign bit to be lost. So although this data format can be used with SCADAWARE, the range is limited to 0-4095 (12 full bits) on the Modbus side, resulting in a SCADAWARE range of 0-32K with a minimum resolution of 8.

The fifth entry at row E is typical for an 8 bit D/A output. The low resolution D/A output device in the PLC or RTU has a range limited to 0-255. Because SCADAWARE uses the 16 bit signed integer format, right shifting will be required to convert the internal 16 bit signed integer into an 8 bit unsigned integer. The MAP file bit size for this entry is 8, causing SCADAWARE to remove the lower 7 bits or resolution from the internal integer format when formatting data

for Modbus. A similar process would occur if data were received from an 8 bit input device, which would have similar restrictions to that of the 12 bit example on row B.

The bit size for all integer registers is reflected in the MAP DUMP output file. Any channels not using integer format will have the word FLOAT in place of the bit size.

Examples of the bit size option are as follows:

```
Map Ireg 10 A1:a16 Value 14 ;receive raw A/D data from oddball 14 bit unit
map Hreg 15 V1:v8 Value   8 ; reduce resolution for 8 bit D/A output card
map Ireg 20 A1:a8 Value     ; use default 12 bit Ireg setting
map Hreg 20 A1:a5           ; use default 15 bit Hreg setting
```

MODBUS DATA TRANSFERS

Once the proper mapping is installed, TSP data can be sent and received over a Modbus link by referencing the Modbus addresses only. A task in SCADAWARE will send and received data with properly formatted commands generated by MAP SCAN and MAP DATA statements. Aside from these statements, data transfer programming is very similar to normal TSP procedures. The main difference is that instead of normal TSP commands for SCAN and DATA, the modbus equivalent MAP SCAN and MAP DATA are used. All other TSP facilities are available just as if Modbus weren't a consideration.

The MAP SCAN command is processed similar to a BLOCK SCAN command in regular TSP. Processing will send a formatted Modbus command to a remote unit asking it for data. The MAP SCAN line must provide the following parameters:

Modbus ID of receiver	
Modbus Command Code	RI, WC, W1C, RHR, etc.
Starting Modbus address	As defined in the MBR file.
Register Count	How many registers (not required for W1C)

A line to read holding 8 registers starting at 4005 from unit 4 would look like:

```
Map Scan 8 RHR 5 4 ; unit 8, read HR, starting at 5, count of 8 registers
```

The receiving unit will process this line and send back a Modbus response. Because the addresses aren't included in most responses, SCADAWARE has to keep track of the last command sent out so that it will know where to put the data. When a response is received, SCADAWARE uses the map table to locate the appropriate TSP channels which will receive the data. The format of the data was determined when the map file was processed, so all that is needed in the scan line is the data shown above. All the details of floats, integers, etc. is handled automatically by SCADAWARE.

The MAP DATA command is used to send data to another unit, much like a BLOCK DATA line is used in regular TSP. The MAP DATA line is similar to the MAP SCAN line in that the programmer must provide the necessary Modbus information with which to build the line. SCADAWARE will format the line using data from the TSP data table as directed by the Modbus map table in use.

A line to write 6 holding registers on unit 7 starting at register 4 looks like:

```
Map Data 7 WHR 4 6 ; unit 7, 6 regs, starting at 4 (4004 in the other unit)
```

The other Modbus commands are very similar. The only difference is with the Write 1 Coil command (W1C)

which does not require a count because only 1 coil can be written.

Note that most values must be specified literally, with a number. The only exception is the unit number which can be any valid TSP expression. This feature is useful when designing a program which must poll a number of identical units, on which the only difference is the unit ID itself. If a variable is used instead of a hard-coded number, a common routine can be used for any unit. As with any other TSP procedure, command line parameters can also be used for this same purpose.

```
Calc x = 10 ; read from unit 10 this time
Map Data X 4 RIR 30 8 ; unit x, read inputs, count of 8, starting at 30
```

SAMPLE MAP FILE

The following file sample contains a variety of options, more than would likely appear in a small system.

```
; Modbus Test File
msg Modbus Definition File Start
units 3                          ; total of 3 modbus rtus to be simulated
Protocol RTU       ; use RTU (and not ASCII mode) for the entire system

Msg Starting first Modbus map for SS180
sele ss180         ; Make all TSP references apply to SS180 RTU
ID  17                          ; first unit is Modbus ID 17
msg Defining Table Sizes
table coil 200           ; output bits on the logical RTU
table input 32           ; input bits
table hreg 32            ; holding registers
table ireg 10            ; input registers
map coil 0 o1            ; Modbus coil 0 is value of TSP output number 1
map coil 1 o1 NEW        ; Coil 1 is new alarms status of output 1
map coil 2 o1 ALARM      ; coil 2 is in alarm status of output 1
map coil 3 o1 DB         ; coil 3 is deadband alarm status for O1
map coil 4 o1 RESET      ; coil 4 is reset status of O1
map coil 5 o1 ABNORMAL   ; coil 5 is abnormal status of O1
map coil 6 o1 NEW ALARM  ; coil 6 is New or In alarm
map coil 7 o1 NEW RESET  ; coil 7 is new or reset status
map coil 8 o2            ; coil 8 is value of O2
map coil 9 o2 NEW
map coil 10 o2 ALARM
map coil 11 o2 DB
map coil 12 o2 RESET
map coil 13 o2 ABNORMAL
map coil 14 s1:s16       ; coils 14-29 are status channels 1-16
map coil 30 v1:v5        ; coils 30-34 are value channels 1-5

msg Setting input Map
map input  0 s1:s16      ; input bits 0-15 are TSP status inputs 1-16
map input  20 o1:o16     ; input bits 20-35 are TSP outputs 1-16

msg Setting IREG map
map ireg 0 a1:a4         ; Input registers 0-9 are TSP Analogs 1-10
map ireg 4 a5:a8 14      ; second board uses 14 bit A/D mode
```

```
msg Setting Hreg Map
map hreg 0 v1:v8 float ; use 2 Hregs to hold 1 float, times 8 points
map hreg 16 a1:a8

Msg Starting Second Modbus RTU definition
ID  2           ; small unit with coils only
table coil 100
map coil 0 s1:s16

Msg Defining Third unit that has ID of 8
ID 8
table coil 50 100 ; 50 spaces starting with address 100
table input 20
map coil 101  s1:s8
map input 10 s1:s16

msg Modbus File Done

msg Starting Modbus Definition File #1
delay 5             ; max of 2 ticks allowed between incoming characters
units 10             ; 3 ROC and 3 Flow computers
tag ROC             ; nickname only

ID 1                        ; PC Platform is Modbus 1
PROTOCOL daniels            ; will talk to Daniel's flow computer
table input  32             ; status inputs starting at 1
sele PC
map input 0 s1:s32          ; Modbus bits 0-x into TSP 1-x

;Flow compute has its own modbus entry
ID 11                       ; Make flow computer 10 more than RTU
protocol swap          ; uses 2 Hregs for floats, swapped from normal
table hreg 16
table input 16
table coil 16
table ireg 16
map hreg 0 v1:v4 value float       ; hold registers 0-x map to A1
map input 0 s1:s8
map coil 0 o1:o8
map ireg 0 a1:a4
map hreg 10 a5:a8 value 12
```

INSTALLING MODBUS FOR SCADAWARE

Modbus is a built-in feature of the DPMI versions of SCADAWARE, RTUMON3 and RTUMED3. Modbus is not included in the Lite Version. Although there is no additional cost for the DPMI version of SCADAWARE, computers and RTU's may require hardware enhancement in order to support in order to operate the larger program.

Aside from the normal SCADAWARE requirements, operation of the Modbus interface requires completion of the following steps:

1 Preparing the Modbus MAP definition file which relates the Modbus addresses to the TSP data values within

SCADAWARE.

2 Modifying the startup process to include loading of the Modbus Map file. This requires one or more MAP LOAD commands in the computer's startup procedure.

3 Use the SCADAWARE Map Dump command to produce a text file containing details on the Modbus data mapping. Make sure that all TSP and Modbus registers are properly sized and aligned.

4 Adding a new RTU task, or modifying an existing RTU task to process the incoming and outgoing Modbus commands. The task does not need to be dedicated to Modbus only. It need only select the appropriate data map when performing Modbus functions.

5 For Modbus Slave Applications: Modifying the startup procedure for the Slave task such that the proper data map is selected. Once selected, the map will remain active until de-selected by that task. For tasks operating as a dedicated slave processor, no additional programming is required because all Modbus commands will be automatically processed by the task.

6 For Modbus Master applications: Preparing or modifying TSP Link procedures to contain MAP SCAN and MAP DATA instead of the usual TSP SCAN and DATA commands. The procedures are similar to normal TSP link files, but use the MAP versions of the data transfer commands.

The following procedure fragments illustrate the modifications required to existing SCADAWARE procedures. Important portions are shown in **bold:**

--- fragment of DAT file ---

```
COM 1 1200 multidrop N 8 2 200 400    ; Radio
com 2 1200 multidrop N 8 2 200 400    ; Modbus interface

msg Setting Up Tasks
variables 32                                        ; vars for task 0
task, rtu, Coml Radio
task, rtu, Com2 PLC          ; locally connected Modbus PLC
task, rtu, Utility Task
task, driver, Local I/O, relay, base.rly                      ; task 5
|
|
```

--- Fragment of task 0 startup ---

```
|
|
read VR130      ; read in VR 130 setup
read SMI10       ; read in SMI 110 setup
sele VR130
map load 1 modbus host.mbr  ; read in the Modbus Map definition into slot 1
task 1 start           ; start up the normal TSP radio task
task 2 start           ; start up the Modbus direct link task
user                     ; put up the user menu for task 0
```

--- fragment of Task 2 Startup for Dedicated Modbus Slave ---

```
proc start2
 set baud 1200
 map sele PLC        ; select map by name instead of number
```

--- small Map File named HOST.MBR --

```
msg Starting Modbus Definition File #1
Protocol RTU        ; use binary protocol for entire MAP
delay 5             ; max of 2 ticks allowed between incoming characters
units 2             ; 1 PLC and 1 Flow Computer
tag PLC             ; nickname only

ID 1                ; PC Platform is Modbus 1
PROTOCOL integer    ; no special treatment for floats required here
table input  32     ; status inputs starting at 1
table coil 32
table hreg 16
sele VR130
map input 0 s1:s32        ; Modbus bits 0-x into TSP 1-x
map hreg 0 v1:v16         ; holding regs go to Value Channels in SCADAWARE
map coil 0 o1:o16         ; Tsp outputs to PLC outputs
;Flow compute has its own modbus entry
ID 11                     ; Make flow computer 10 more than RTU
protocol daniels
table hreg 16
table input 16
table coil 16
table ireg 16
map hreg 0 v1:v4 value float       ; hold registers 0-x map to A1
map input 0 s1:s8
map coil 0 o1:o8
map ireg 0 a1:a4
map hreg 10 a5:a8 value 12
```

--- fragment of Link file for Modbus Master. It polls 2 units---

```
Proc link5
sele VR130
map sele PLC
map scan  1 RI  1 32      ; read all inputs from the PLC in one command
map data 1 WC 1 16    ; write outputs to the PLC
map data 1 W1C 2         ; note that no count is required for W1C
map scan 1 RHR 1 16    ; read all Holding registers

; note that the next lines use address 11 for the Modbus address
map scan 11 RIR 1 16    ; read first 16 input regs
map scan 11 RIR 17 16 ; read next 16 input regs
link reset      ; clear the TSP link if active
bye         ; force task offline to allow next link to process quickly
```

——— END OF SCADAWARE MODBUS INTERFACE MANUAL ———

AM2/tt 81220