TOTAL ENGINEERING SERVICES TEAM, INC.

RTU / SCADA SYSTEMS

MODBUS INTERFACE MANUAL

Document No. 1220-01

Sep 20, 1993

P.O. Drawer 1760
671 Whitney Ave.
Gretna, La. 70054
(504) 368-6792

VOID

1 - INTRODUCTION

This document provides detailed information on TEST's implementation of the MODBUS RTU Communications Protocol. This software interface allows TEST's SCADAWARE™ program to communicate in an *OPEN ARCHITECTURE* system. Normally, TEST SCADA systems use a special communications language called TEST SCADA PROTOCOL. TSP provides a number of high-level mechanisms for controlling remote devices and exchanging a wide variety of SCADA type information. However, TSP is proprietary to TEST systems and cannot be used on equipment provided by other vendors. The TEST Modbus interface provides a standardized method of integrating the advantages of TEST's system with otherwise incompatible devices from other manufacturers.

The MODBUS interface can be used to connect a complete TEST SCADA system to a completely separate system such as a Distributed Control System (DCS) or to a Programmable Logic Controller (PLC). The interface can also be used to let the standard SCADAWARE™ program connect with RTU's that work with the standard Modbus protocol. This allows new or existing RTU equipment to take advantage of TEST's SCADAWARE™ software.

This document focuses on the TEST MODBUS interface and how it is implemented on a SCADAWARE™ based computer. Please refer to companion document *TEST SCADA PROTOCOL TUTORIAL* for more information on TEST's proprietary protocol and why it is a good choice for many system functions, including those requiring an Open Architecture design.

2 - OPEN ARCHITECTURE

The term *Open Architecture* has begun to appear often in SCADA system documents and specifications. 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, including those from different vendors. Unfortunately, this concept is much easier to specify than it is to implement. Contrary to what is often stated in the press, 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.

3 - SCADA PROTOCOL BASICS

TEST's SCADA system, like any other SCADA or telemetry system, uses a precise method of exchanging data and commands among the units connected to the network of devices. 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 that is designed to best handle the functions actually required for this type of equipment in typical remote SCADA applications. Admittedly, this protocol is quite a bit different from the traditional 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 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.

4 - 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 developed since then into a complex language of its own that is ideally suited to modern 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.

A separate document (TEST PROTOCOL TUTORIAL) explains the many benefits of TSP in real-life installations. These include features that make the system faster, more flexible, and more reliable than any comparable protocol in a similar situation. We are obviously very proud of our systems and the success they have achieved all over the world. We can modestly say that TSP has only one minor flaw: it's not directly compatible with other protocols.

5 - MODBUS AND OPEN ARCHITECTURE

Although there is no true 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 implementation is a very complete and comprehensive MODBUS protocol 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 allows a master unit to send and receive binary data with slave units 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 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.

6 - 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.

7 - TEST'S MODBUS INTERFACE

MODBUS defines only low level data types such as bits and words. TSP provides high-level data types such as Analog Values and AGA-3 gas flow meter calculations. The interface provided by TEST allows all of the important TSP data types to be easily "mapped" into equivalent low-level Modbus coils, inputs, and registers for transport to external systems.

TSP DATA TYPES	MODBUS DATA TYPES
Status Inputs & Outputs	Bits
Analog, Value, PID, AGA-3,	Words
Timer, Counter, Special Function,	IEEE Floating Point
Totalizer, Low Alarm, Hi Alarm,	
Alarm Delay, Deadband, Alarm Times,	
Counter/Totalizer Reset Time,	
Current Alarm mode.	

Built-in TSP Features such as alarm status, deadbands, setpoints, and other internal system variables are easily accessed on the Modbus link. The relatively low point count limitations of Modbus are overcome by letting a single TSP/Modbus interface simulate up to 16 separate Modbus RTU addresses. This allows thousands of equivalent Modbus values to be transferred efficiently and quickly using standard Modbus data transfer commands.

8 - TSP DATA MAPPING

The data map configuration provides many options for selecting which TSP data points appear in the logical Modbus memory. All Modbus data accesses occur at numeric addresses. These addresses begin at 0 and run up to a limit determined by each separate piece of hardware. For example, a small PLC may only allow 32 input points which would be numbered as Modbus inputs 0-31. A Modbus protocol data transfer would specify an Input point transfer beginning at a certain address and running for some number of points. The actual meaning of the data points is irrelevant to Modbus.

The Data Mapping process allows assignment of TSP values and channel status conditions to these imaginary Modbus points. For example, we may assign the actual value (off or on) 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 designed.

9 - MODBUS MAP CONFIGURATION

The system designer must decide how the TEST RTU data will be mapped into the equivalent MODBUS data types. Although the Modbus choices are limited to bits, words, and floats, TEST's Modbus system allows a convenient and flexible way to let 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 value 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 of from where it came.

Data Mapping requires some planning to insure that the proper TSP data is available in a form convenient to the external system. This is normally required in any Modbus type system because there is no built-in capability to do all of the things automatically done by a TEST 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 systems' needs for the TSP data.

The configuration file can have any file name, although one with the file type of "MBR" (for ModBus RTU) is recommended. 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.

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 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.

10 - SETTING UP A MODBUS TASK

Setting up a task for Modbus communications involves a simple 3 step process. These steps are:

1. Define the task
2. Load the map file
3. Select the data map

STEP 1. Any RTU type task, except task 0, can be used for Modbus communications. A single line in the main configuration (DAT) file is all that is needed to define an RTU type task. Examples of such lines are:

```
TASK RTU Com1
TASK RTU Modbus
```

The keyword TASK indicates a new task definition and the keyword RTU indicates the type of task. The third item on the line can be up to 16 characters long and is simply the task name.

STEP 2. In order for the TEST SCADA system to interface with another system using the Modbus protocol, a file must be processed which "maps" TEST SCADA data types into Modbus data types. The details of what is contained in the map file are described elsewhere in this document. For now, we assume that the map

file has already been created and that it is correct.

The MAP LOAD command must be used to process the map file and setup the mapping scheme. This command can be processed by any RTU type task, including task 0. Normally this command is processed by task 0 during program startup. This can be either in the START0.RTU file or a subroutine called from this file. Again, the details of the MAP LOAD command can be found elsewhere in this document in a section called MAP COMMAND.

STEP 3. Once a mapping scheme has been loaded it is accessible by any RTU type task. However, the default protocol for all RTU type tasks is the TSP protocol. In order for a task to access a data map and use the Modbus protocol the MAP SELECT command must be used. This command is usually processed from within the STARTx.RTU file which automatically get processed when the task is started. Once a valid data map is selected for an RTU type task, the protocol for that task is determined by the type of map selected (so far Modbus is only type available). For more details refer to the MAP SELECT command in the section called MAP COMMAND.

11 - MAP FILE KEYWORDS

MSG — Display a text message on the system console.

DELAY — Set the maximum number of ticks allowed between incoming characters. The default value is 2 system ticks. 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.

TAG — Set the tag name for the data map being defined. By default, the tag names for each data map are MAP1, MAP2, MAP3, and so on. These tag names can be used by any RTU type task to select a data map by tag name rather than by index number.

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

UNITS — Within each data map, up to 16 Modbus IDs can be defined, each with its own map setup. 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 1 is assumed.

ID — Specify the ID number to be used for subsequent data maps. No table sizes or data maps can be defined until an ID has been specified. Each ID command will begin the next map setup.

TABLE — Specify the size of the modbus map table for specific Modbus data types. Parameters include a keyword specifying the type of table being defined and a number specifying the size of the table. 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.

MAP

Specify a link between TSP values and 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. Remember, Modbus tables are offset by 1. This means that Modbus tables start at position 0 and extend up to the maximum table size minus 1.

The next parameter 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 are strictly optional. These parameters are keywords which specify exactly what data pertaining to the 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.

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 a channel 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.

VALUE	- if current value of channel < > 0
NEW	- if in alarm but not yet acknowledged
ALARM	- if acknowledged but still in alarm state
RES	- if no longer in alarm and waiting for reset
ABNORMAL	- 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
MAP Coil 1 O1 New
MAP Coil 2 O1 Value New
```

Now consider the value of Modbus Coil locations 0, 1, and 2 after 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
```

Now consider the state of the TEST SCADA unit 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 does not contain a 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.

There is one last point worth mentioning about the mapping of Modbus Coils and Inputs. Since these locations 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.

For Modbus **Input Registers** and **Holding Registers**, each location contains a 16 bit value. What each 16 bit value 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.

VALUE	- integer representation of channel's current value
HI	- hi alarm setpoint
LO	- lo alarm setpoint
DB	- deadband value
FLOAT	- floating point representation of a channel's current value, hi alarm setpoint, lo alarm setpoint, or deadband value

If no keyword is specified the keyword VALUE is assumed. By default, whenever a channel value, hi or lo alarm setpoint, or a deadband value is mapped to a register location it is represented by a 16-bit integer value and occupies a single register. As an alternative, the FLOAT keyword can be specified in conjunction with any of the other keywords to have each channel mapping represented by a 32-bit floating point number. This 32-bit number will occupy 2 consecutive 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. For example, consider the following map setup line.

```
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.

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	1 - upper 2 bytes
1	1 - lower 2 bytes
2	2 - upper 2 bytes
3	2 - lower 2 bytes
4	3 - upper 2 bytes
5	3 - lower 2 bytes
6	4 - upper 2 bytes
7	4 - lower 2 bytes

If an attempt is made to read or write any registers starting with a location containing the lower 2 bytes of a floating point number, an automatic adjustment would be made to skip that location and start with the next location in the specified range.

When setting up a map for registers, only one of the keywords VALUE, HI, LO, and DB should be specified at a time. When a MAP line is processed a check is done to see if two keywords are specified. Any keywords beyond two are ignored. 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:

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

12 - SAMPLE MAP FILE

```
; Modbus Test File
msg Modbus Definition File Start
units 3                          ; total of 3 modbus rtus to be simulated
msg Selecting SS180
sele ss180                       ; default to SS180 RTU
msg Starting Unit 1

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 16                    ; holding registers
table ireg 10                    ; input registers
msg Setting Coil Map

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:a10                ; Input registers 0-9 are TSP Analogs 1-10

msg Setting Hreg Map
map hreg 0 v1:v8
map hreg 8 a1:a8

ID  2
table coil 100
map coil 0 s1:s16

ID 3
table coil 100
table input 20
map coil 0  s1:s8
map input 10 s1:s16
msg Modbus File Done
```

13 - MAP COMMAND

This command is used to load a map file, select a particular data map for a task, send data to and receive data from another unit using the Modbus protocol, and list the configuration of a data map to a file. The MAP command can be processed only by RTU type tasks. The format of this command is the word MAP followed by a keyword and some additional parameters. The available keywords and an explanation of each are given below.

MAP SELECT xx

Up to 10 data mapping schemes can be defined on a TEST SCADA system at one time. However, each task can have access to only one data map at a time. This command is used to select a particular data map for an RTU type task. A data map can be selected by index number or tag name. Valid index numbers range from 1 to 10 and the default tag name of each map is the keyword "TAG' followed by the index number. This tag name can be changed from within the map file during a map load.

By default, the current data map index for each task is 0. This means that there is no default data mapping scheme for any task and the default protocol for all RTU type tasks is the TSP protocol. Once a valid data map is selected for an RTU type task, the protocol for that task is determined by the type of map selected (so far Modbus is only type available).

The command MAP SELECT 0 can be used by an RTU type task to disassociate itself with any data map. This will cause the protocol used by that task to be returned to the TSP protocol. If a third parameter is not specified in the command, a message will be sent to the task processing the command to show the current map index for that task.

MAP LOAD xx MODBUS [filename]

This command is used to process a map file which defines a data map. In this command a valid index number in the range 1 - 10 must be specified in place of the xx shown above. Specifying a map tag name instead of an index number will not work for this command. Because the data map to be defined is specified in this command by an index number, the task processing the command does not have to select this map, or any data map, prior to processing this command.

Following the map index number is a keyword which specifies the type of map being defined (MODBUS is only available map type at this time). The last parameter on the line is optional and is used to specify the name of the map file to be processed. If no file name is specified, the default file name RTU.MBR is assumed. If a file name is specified but does not contain an extension, the default file extension MBR is assumed.

If the map file does not exist, the command will be ignored and the current data map for the specified map index will remain unchanged. Otherwise, the current data map for the specified map index will be cleared out and the file will be processed to define the new map configuration.

MAP DUMP [filename]

This command is used to list the configuration of a data map to a file. This command can be processed by any RTU type task, but two conditions must be met prior to processing the command if it is to be successful. First, the

task processing the command must have a valid map selected. Second, the currently selected map must be defined from a prior MAP LOAD command.

The last parameter on the line is optional and is used to specify the name of the file to contain the listing of the map configuration. If no file name is specified, the default file name MODBUS.MAP is assumed. If a file name is specified but does not contain an extension, the default file extension MAP is assumed.

MAP SCAN id cmd start count

This command is used to request data from a remote unit using the modbus protocol. This command can be processed by any RTU type task except task 0. Before processing this command, the task should have a valid data map selected which has already been defined using the MAP LOAD command.

Within each data map, up to 16 Modbus IDs can be defined each with its own map setup. The number of Modbus IDs contained in any data map is determined by the UNITS command which gets processed from within the map file during a map load. By default, all Modbus ID names are set to 0. These IDs can be changed by using the ID command in the map file.

When using the SCAN command, a valid Modbus ID from the currently selected data map must be specified. This determines which unit will respond to the outgoing message and where the incoming response will be placed.

Following the Modbus ID is a keyword which specifies the type of mapped data being requested. The available keywords and their purposes are as follows:

RC - Read Coils
RI - Read Inputs
RHR - Read Holding Registers
RIR - Read Input Registers

The last two parameters that must be specified in the SCAN command are the starting location and number of locations within the map for which data is requested. All mapping starts at location 0. For example, if a table of 200 coils is defined, the actual locations that exist are 0 - 199. If the start location specified is not within the valid range an error will result. If the number of locations specified extends beyond the upper limit, the count will automatically be reduced to eliminate the overrun.

MAP DATA id cmd start count

This command is used to send data from one unit to a remote unit using the modbus protocol. The specifics of this command are very similar to the MAP SCAN command. This command can be processed by any RTU type task except task 0. Before processing this command, the task should have a valid data map selected which has already been defined using the MAP LOAD command.

Within each data map, up to 16 Modbus IDs can be defined each with its own map setup. The number of Modbus IDs contained in any data map is determined by the UNITS command which gets processed from within the

map file during a map load. By default, all Modbus ID names are set to 0. These IDs can be changed by using the ID command in the map file.

When using the DATA command, a valid Modbus ID from the currently selected data map must be specified. This determines where the outgoing data will come from and which unit will receive the outgoing data.

Following the Modbus ID is a keyword which specifies the type of mapped data being sent. The available keywords and their purposes are as follows:

WC - Write Coils
WHR - Write Holding Registers

The last two parameters that must be specified in the DATA command are the starting location and number of locations within the map for which data is being sent. All mapping starts at location 0. For example, if a table of 200 coils is defined, the actual locations that exist are 0 - 199. If the start location specified is not within the valid range an error will result. If the number of locations specified extends beyond the upper limit, the count will automatically be reduced to eliminate the overrun.

END OF OPEN ARCHITECTURE MANUAL

AMZ/nt MODBUS.DOC