

Using the Universal Gateway Modbus TCP with an S7-1200 PLC (v8)

Table of Contents

1. Overview
2. Add Modbus TCP Device
3. Add Device Tags
 - a. Add \$User Device Tags
 - b. Add Modbus TCP Device Tags
4. Create First Tag Map
5. MB_Server code in the S7-1200
6. Test First Tag Map
7. Modbus Addressing & Memory Areas

Appendix A: How to read STRING data type over Modbus TCP from S7-1200

Appendix B: Tag copying results between different data types

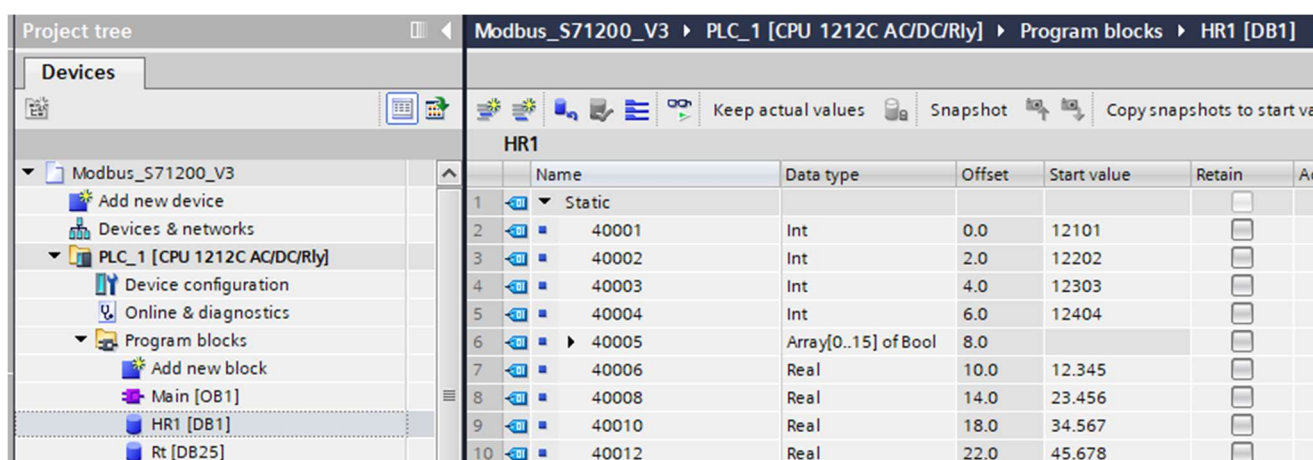
Appendix C: Data type Value Ranges

Appendix D: Troubleshooting Modbus TCP with The Trace Manager

Section 1: Overview

This is a brief description of setting up Modbus TCP communications between a Universal Gateway and a Siemens S7-1200 PLC so that the Gateway can read from, and write to, tags in the S7-1200 PLC.

The example program in the S7-1200 requires a variety of tags to be exchanged with a Rockwell CompactLogix PLC. Most of the tags to be exchanged will reside in a Global DB, but tags in the process image area (tag table) of type I, Q and M will also need to be accessed. The tags are a range of data types, including INT, BOOL and REAL.



	Name	Data type	Offset	Start value	Retain	Access
1	Static					
2	40001	Int	0.0	12101	<input type="checkbox"/>	
3	40002	Int	2.0	12202	<input type="checkbox"/>	
4	40003	Int	4.0	12303	<input type="checkbox"/>	
5	40004	Int	6.0	12404	<input type="checkbox"/>	
6	40005	Array[0..15] of Bool	8.0		<input type="checkbox"/>	
7	40006	Real	10.0	12.345	<input type="checkbox"/>	
8	40008	Real	14.0	23.456	<input type="checkbox"/>	
9	40010	Real	18.0	34.567	<input type="checkbox"/>	
10	40012	Real	22.0	45.678	<input type="checkbox"/>	

Figure 1A

The elements in this example are named for the Modbus Holding Register that they are associated with (see Section 7) for ease of illustration. The INT and REAL elements have a value in the Start Value column, to illustrate that no code is required to put understandable data into the registers.

It is straightforward to mix data types in the Holding Registers, but pay attention to the offset required for each data type (see Figure 1A). The offset will point to what the Modbus address is for each register.

- The INT data type is 16 bits which requires a 2 byte offset.
- One WORD arranged as an array of 16 BOOLS is 16 bits which requires a 2 byte offset.
- The REAL data type is 32 bits which requires a 4 byte offset.

No communication code is required in the CompactLogix. Only the single MB_SERVER Instruction is required in the S7-1200. This illustration uses the v3.1 firmware version of the MB_SERVER Instruction.

The S7-1200 needs to be configured:

- If Holding Registers are to be used (the range of 40001 to 49999), a Global DB (this example has a DB named HR1) needs to be added to the PLC and configured.
- A Global DB must have the *Optimized Block Access* checkbox cleared.
 - Optimizing the block access will not allow external devices access to the data.
 - To get to a Data Blocks properties dialog box, right click on it and select Properties (all the way at the bottom of the menu that pops up).

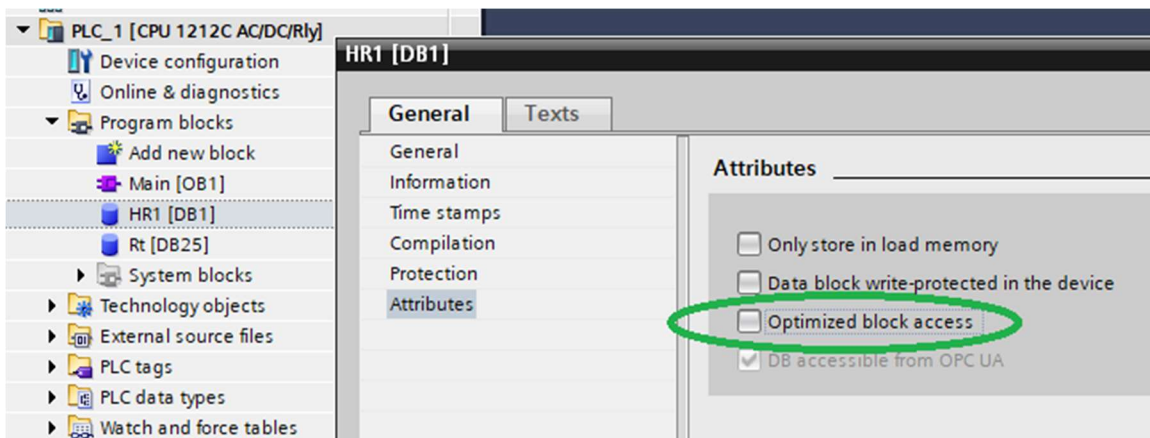


Figure 1B

Note:

When first setting up a Universal Gateway, configure the Time Setup information for your local time. Having the correct time makes troubleshooting easier if you have to look at the error or engineering logs. Here is panel 4 in the Quick Start guide:

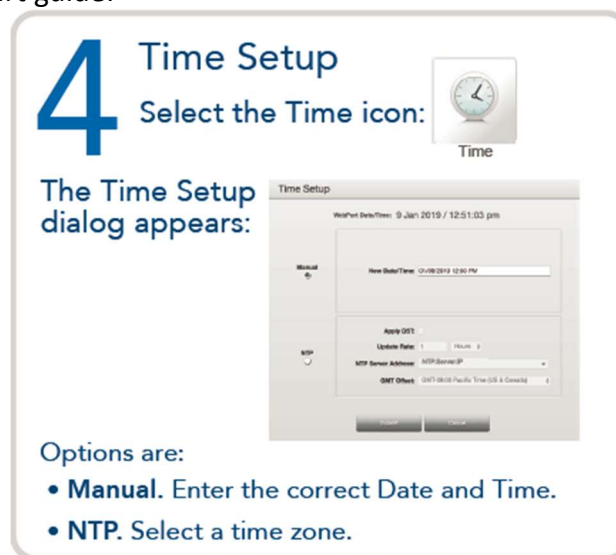


Figure 1C

Alternatively, see section 3.8 *Configuring Time Setup* in the Software Users Guide.

Section 2: Add Modbus TCP Device

The Modbus TCP device to add is a server at the expected IP address of the S7-1200 processor. A Modbus TCP server will respond to requests from a client either with data or error information, but will not initiate requests. The client is a master, and servers are slaves.

Modbus TCP port default is 502 and the slave ID default is 1. The port and ID in the Universal Gateway need to match what is set up in the S7-1200.

Here's what the Device setup looks like.

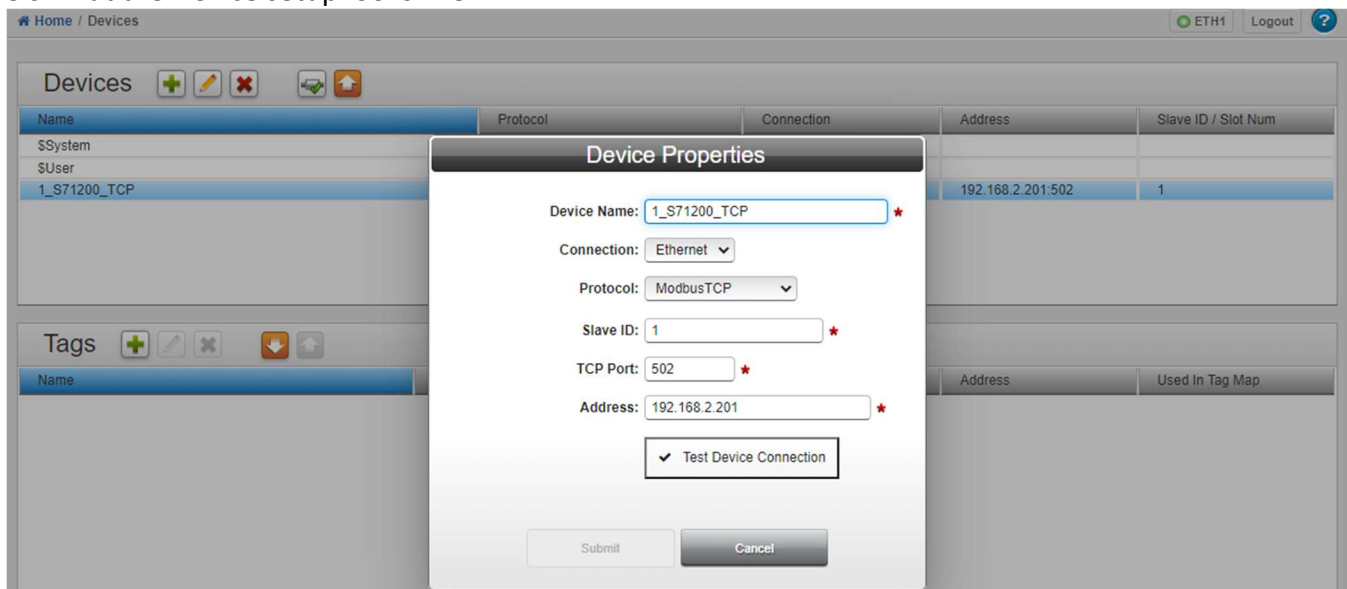


Figure 2

If the Test Device Connection button is pressed right now, the test would fail even if the PLC is online at the correct address, since there is not yet Modbus Server code in the PLC.

Section 3: Add Device Tags

First, open the \$User device and add the following \$User Device Tags.

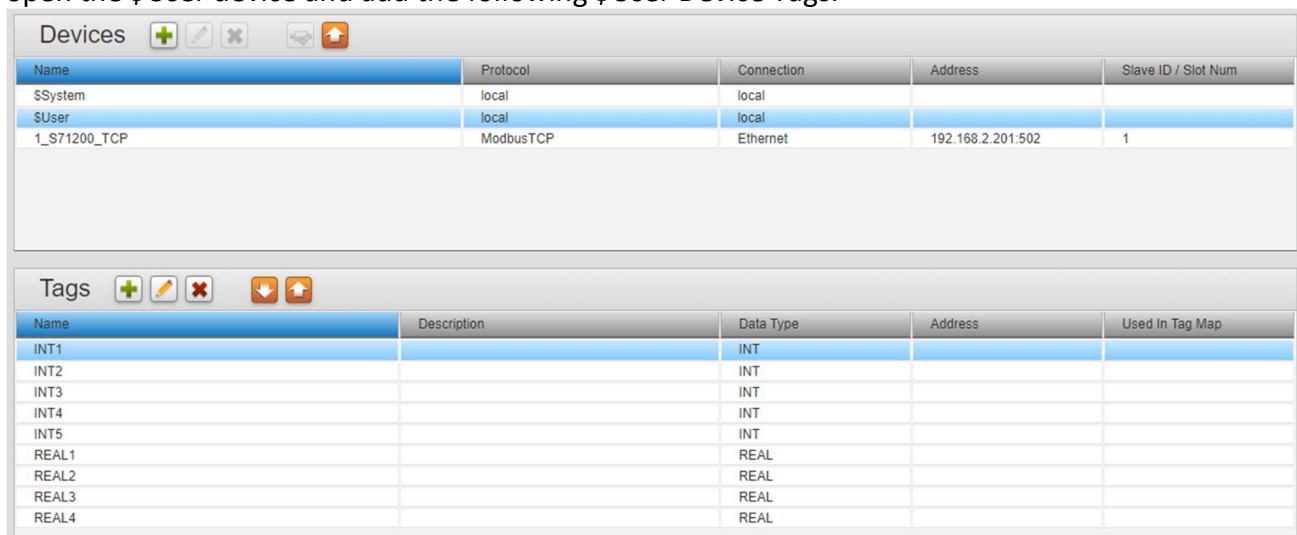


Figure 3A

\$User tags are useful as an uncomplicated way to verify that data is being read from the Device Tags correctly, since there are only the Gateway and the Device to deal with. Once data read is known to be functioning, data writing to the Device Tags will go much smoother.

Take a look at the name column of the tags for the S7-1200 initial test in Figure 1A. There are 4 INTs, a BOOL array (40005) and 4 REAL registers. Here, there are 5 INTs and 4 REALs to get started with.

Next, open the 1_S71200_TCP Device and add the following tags.

Devices				
Name	Protocol	Connection	Address	Slave ID / Slot Num
\$System	local	local		
\$User	local	local		
1_S71200_TCP	ModbusTCP	Ethernet	192.168.2.201:502	1

Tags				
Name	Description	Data Type	Address	Used In Tag Map
40001		SHORT	40001	
40002		SHORT	40002	
40003		SHORT	40003	
40004		SHORT	40004	
40005		SHORT	40005	
40006		REAL	40006	
40008		REAL	40008	
40010		REAL	40010	
40012		REAL	40012	

Figure 3B

The allowable selections for data type of a Modbus TCP device on the Universal Gateway will be explained as this illustration moves along.

Data Type: SHORT ▼

Address:

Byte Swap: ☐ ☐

Is Array ☐

BOOL
 USHORT
 SHORT
 UDINT
 DINT
 REAL
 STRING

Figure 3C

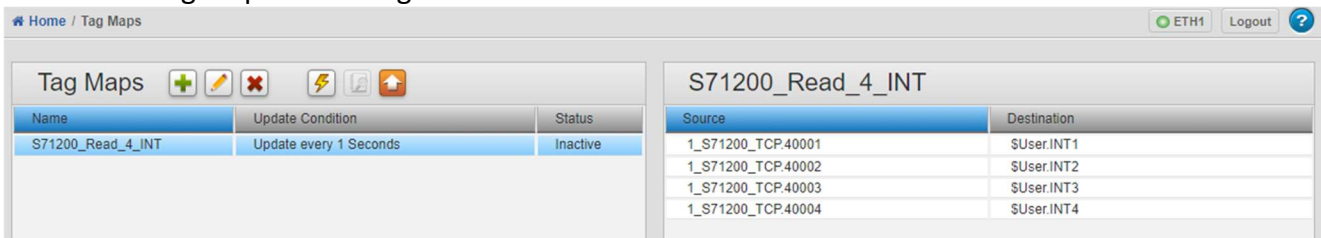
Section 4: Create First Tag Map

The next step is to create tag maps. A Tag Map is a way of reading the data in registers in a Source device and writing them to a Destination device. The data in registers is considered the payload. The payload is extracted from the Source tag using the Source Device protocol and delivered to memory for transmission to the destination tag using the Destination Device protocol. The Source and Destination tags do not necessarily need to have the exact same data type.

Refer to Appendix A: tag copying results between different data types, for more information.

The \$User tags allow data from the PLC to be read and displayed by the Universal Gateway for inspection – no other device is needed. It will be immediately understood if something is wrong with the communications or the data.

This is what the tag map for reading the value of the 4 INTs in the S7-1200 looks like:



Name	Update Condition	Status
S71200_Read_4_INT	Update every 1 Seconds	Inactive

S71200_Read_4_INT	
Source	Destination
1_S71200_TCP.40001	\$User.INT1
1_S71200_TCP.40002	\$User.INT2
1_S71200_TCP.40003	\$User.INT3
1_S71200_TCP.40004	\$User.INT4

Figure 4

A meaningful name for the map should be used. The Update Condition on this example has been set to once every second, which is sufficient to be able to see the data change. When first entered, the Tag Map status is Inactive.

Section 5: MB_Server code in the S7-1200

It's time to examine the code needed for turning the S7-1200 into a Modbus TCP Server.

The MB_SERVER v3.1 Instruction can be found in the Instructions/Communication/Others/Modbus TCP folder in TIA Portal.



Figure 5A

Here's a view of the code (ladder view):

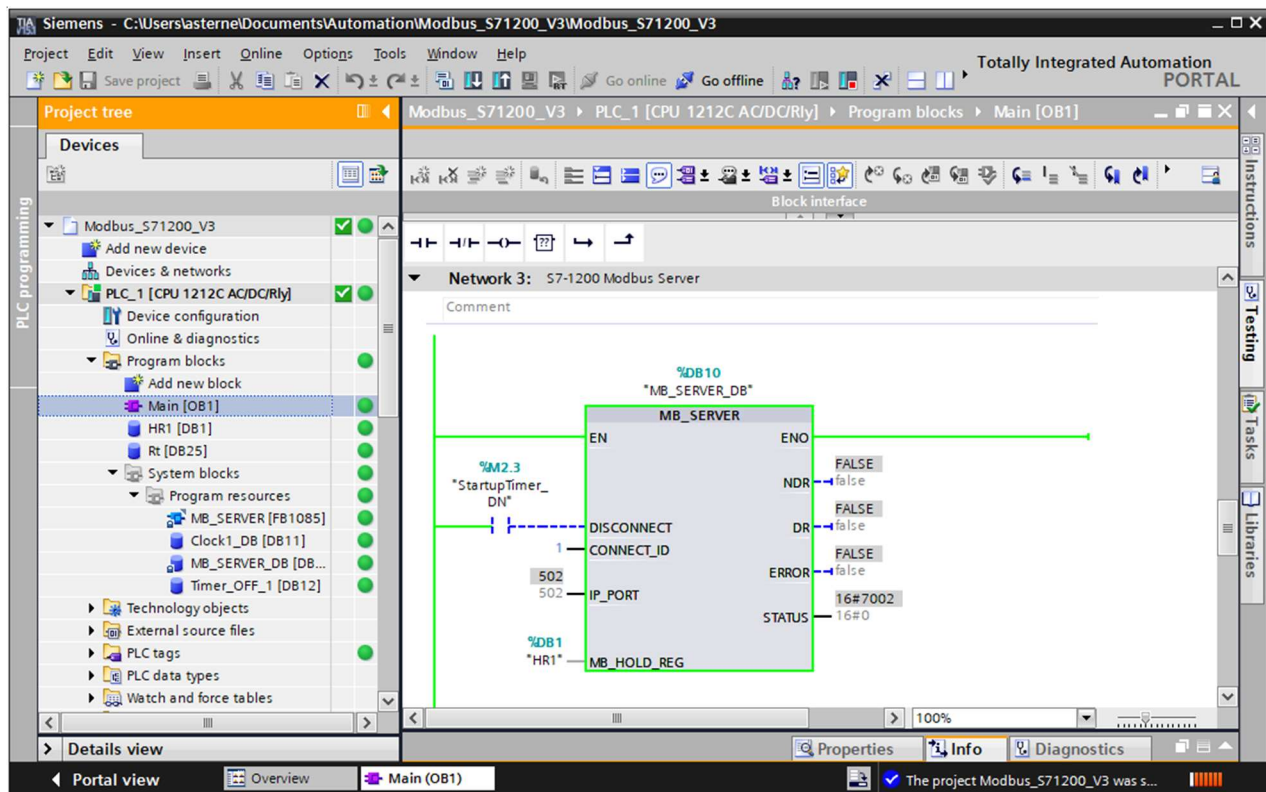


Figure 5B

Here's a description of the required elements of the MB_SERVER v3.1 Instruction starting from the top and moving down the left side, then the right.

When you drag the instruction into the ladder network, you need to assign it a data block, with a unique name and number. In Figure 5B, the name and number of the data block is "MB_SERVER_DB" DB10.

The **EN** input needs to be connected to the power rail.

The **DISCONNECT** parameter controls when a connection request is accepted. If the input is set (true), no operations are executed. The value 7003 is output at the STATUS parameter after a successful connection termination. If the input is reset (false), a passive connection is established. In this example, the tag "StartupTimer_DN" is set for the first 10 seconds after the "FirstScan" bit is reset, which controls when the server starts allowing communication.

The **CONNECT_ID** parameter is similar to a node number. The PLC is capable of supporting multiple instances of MB_SERVER instructions. Each instance of the MB_SERVER instruction in an individual PLC must have a unique CONNECT_ID parameter.

The **IP_PORT** parameter defines which IP port is monitored for connection requests. This value starts at 502 but pay attention to the Information System entry titled *Description of MB_SERVER* for the list of TCP port numbers that must **not** be used. There are several instances of documentation on the Siemens about using multiple ports. See [FAQ Entry ID: 34010717](#) for one.

The **MB_HOLD_REGISTER** parameter is the big one. This points to where the Holding Registers are (40001 to max defined register). The Holding Registers are used for Modbus functions 3 (read Word), 6 (write Word), and 16 (write multiple Words). There are a variety of ways to configure this pointer, see the Information System entry titled *MB_SERVER example: Multiple TCP connections*. Pay attention to how this parameter needs to be entered – in this example, you would enter **%DB1**, NOT "HR1". You need to point it to the Global DB in its entirety, not the first element of the DB.

If needed, the **ENO** can be used to energize coils or instructions as in any other network.

The **NDR** parameter is for "New Data Ready" and is an optional status bit where reset (false) is for no new data and set (true) is for when new data has been written by the client.

The **DR** parameter is for "Data Read" and is an optional status bit where reset (false) is for no data has been read and set (true) is for when data has been read by the client.

The **ERROR** parameter is an optional status bit where set (true) is for when an error occurs, the **STATUS** parameter will hold more information.

The **STATUS** parameter holds the error code. See the Information System entry titled *Description of MB_SERVER* for the hyperlink to the list of STATUS error codes. Pay attention to the section *Parameter STATUS (general status information) (library version V3.x)*, as there is also a list for earlier versions of the instruction. The error codes are separated out into the categories of *general status information*, *protocol error* and *parameter error*. Status values of 16#7002 and 16#0000 are normal. A status value of **16#818C** means the data block being pointed to by the **MB_HOLD_REGISTER** parameter is *optimized*, and will not function since the MB_SERVER instruction requires standard access to data blocks.

Now that the S7-1200 is all set up and in run, when the Test Device Connection button in the Universal Gateway is pressed the test succeeds, and the indication is that a PLC scan is available.

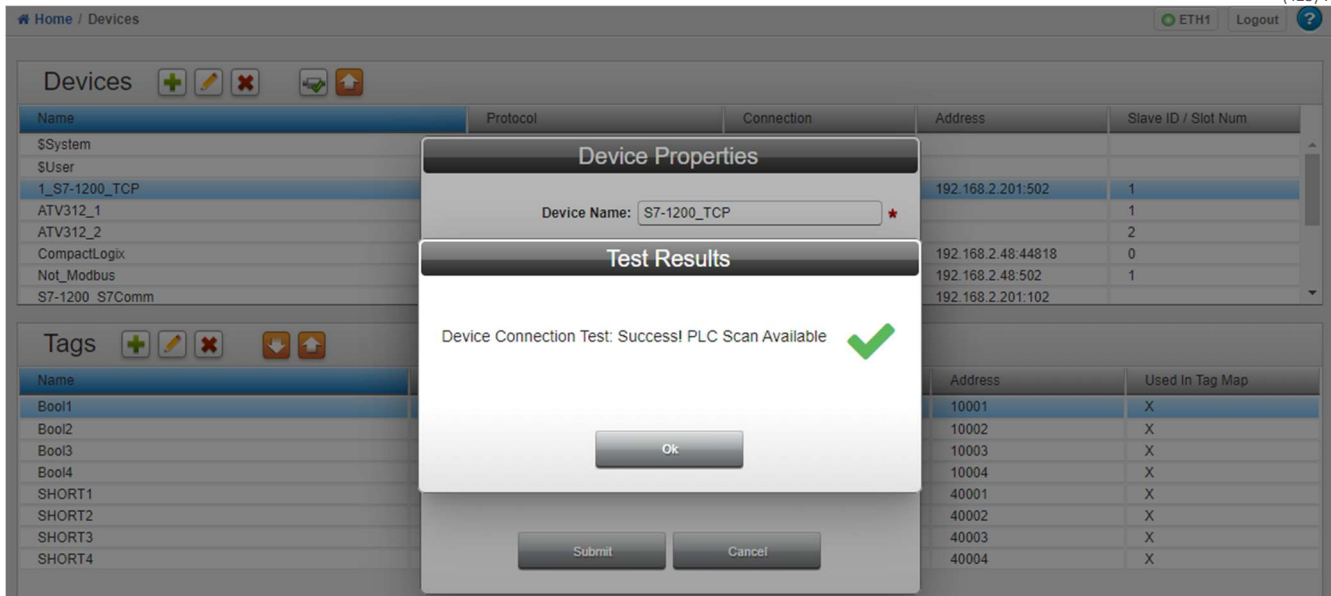


Figure 5C

Section 6: Test First Tag Map

Now that there is Modbus TCP server code in the PLC, it's time to test the communications between the PLC and the Universal Gateway. In the Universal Gateway, navigate to the Tag Maps screen, select the tag map and press the lightning bolt ⚡ button to activate the map. The status of the map will change to Active. Press the View Live Tag Maps 📄 button and the Live Tag Map Viewer will be brought up.

There is more information about how to use the Live Tag Map Viewer in the manual and the in the Gateway online help file (press the ? icon on any page for context sensitive help). In general, the Live Tag Map Viewer allows you to view live tag data values on source and destination tags in a tag map. Tags that have not been read or written use three dash characters (---) to show no values at present. If the source tag can't be read, both source and destination values show as {ERR}. If the destination tag can't be written, only the destination value will be shown as {ERR}.

When the first example tag map is running normally here's what the data looks like in the Live Tag Map Viewer:



Figure 6

Section 7: Modbus Addressing & Memory Areas

Table 1 shows how Modbus TCP tags should be addressed in the Universal Gateway. Initially in this example, the data types set up for the first part of the test are of the Holding Register memory area with Modbus addresses in the 4xxxx range.

Memory area	Modbus Address of the memory area in the device
Coils (output bits) Read/Write	1 to 9999
Discrete Inputs (input bits) Read Only	10001 to 19999
Input Register (input words) Read Only	30001 to 39999
Holding Register (output words) Read/Write	40001 to 49999 400001 to 465536 with extended address area

Table 1

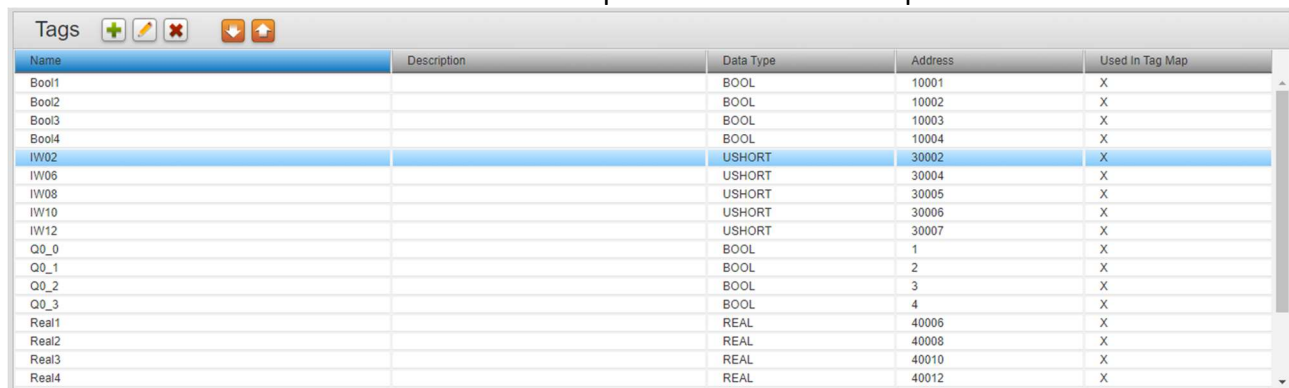
For the Holding Registers in the S7-1200, you can mix and match any of the allowable data types of a Modbus TCP device on the Universal Gateway. See Figure 3C and Appendix B for how some of the data types of the S7-1200 will be mapped into the data types available in the Universal Gateway for a Modbus TCP device.

To read PLC *binary inputs*, it is required to use the Modbus Address of the **Discrete Inputs** (input bits) memory area in the device from Table 1. Similarly, to read PLC *analog input* addresses it is required to use the Modbus Address of the **Input Register** (input words) memory area in the device from Table 1.

Look at the address column of the tags in Figure 7B.

Discrete Inputs: Bool1 – Bool4 correspond to Modbus addresses 10001 – 10004 and are the first 4 physical binary inputs addressed in the S7-1200 as %I0.0 through %I0.3.

Input Registers: IW02 – IW12 correspond to Modbus addresses 30002 – 30007 and are the first 4 physical analog inputs addressed in the S7-1200 as %IW2 through %IW12. Why does the Modbus address start at 30002 and not 30001? Because 30001 is the same as %IW0 which contains as the first 4 bits the addresses %I0.0 through %I0.3. You could read the entire word %IW0 as Modbus address 30001 and pass that on to a destination PLC but that one would need to break up the word into its component bits.



Name	Description	Data Type	Address	Used in Tag Map
Bool1		BOOL	10001	X
Bool2		BOOL	10002	X
Bool3		BOOL	10003	X
Bool4		BOOL	10004	X
IW02		USHORT	30002	X
IW06		USHORT	30004	X
IW08		USHORT	30005	X
IW10		USHORT	30006	X
IW12		USHORT	30007	X
Q0_0		BOOL	1	X
Q0_1		BOOL	2	X
Q0_2		BOOL	3	X
Q0_3		BOOL	4	X
Real1		REAL	40006	X
Real2		REAL	40008	X
Real3		REAL	40010	X
Real4		REAL	40012	X

Figure 7B

This example PLC (1212C AC/DC/RLY) has 2 analog inputs on board, and there are an additional 4 analog inputs on a local module SM 1231 AI (AI 4x13BIT). The first on board analog input has been assigned the address IW2 (the second on board analog input, IW4 is a spare), then the next 4 analog inputs are on the SM 1231 and have been assigned the addresses IW6 – IW12. These *input words* correspond to Modbus addresses 30002 and 30004 – 30007. The word IW0 (30001) corresponds in this example to the first PLC input word, whose bits consist of the first discrete inputs, and the first 4 are addressed in Figure 7B as address 10001 -

10004. So, 10001-10004 are the first 4 bits of 30001. In the PLC, these discrete bits are addressed as %IW0.0 to %IW0.3.

By adding a CompactLogix to the Universal Gateway as a device, a tag map can be made where the CompactLogix can control physical outputs in the S7-1200. In Figure 7B, the first 4 outputs are named Q0_0 to Q0_3, and given the Modbus addresses 1 – 4 (the Coils memory area) which correspond to %Q0.0 through %Q0.3 in the S7-1200.

Appendix A:

How to read STRING data type over Modbus TCP from S7-1200

When needing to transmit STRING data type from a Siemens S7-1200 PLC using Modbus TCP, you need to remember several things:

- Modbus Holding Registers are always transmitted as 16 bit integer.
- You need to convert the HEX value for two characters into one decimal number.

Example: If you wanted to transmit a STRING of characters “ABCD” you need 2 INT’s (for example 40014 and 40015) with 40014 being stuffed with “AB” and 40015 being stuffed with “Cd”.

Here’s the chart for the 4 characters:

Hex	Character
41	A
42	B
43	C
64	d

The HEX value for “A” is “41” and for “B” is “42”. The value entered into 40014 needs to be one decimal number so you need to convert HEX 4142 into its decimal equivalent.

HEX 4142 = Decimal 16706 so 16706 is entered into 40014.

The HEX value for “C” is “43” and for “d” is “64”. The value entered into 40015 needs to be one decimal number so you need to convert HEX 4364 into its decimal equivalent.

HEX 4364 = Decimal 17252 so 17252 is entered into 40015.

In summary, to transmit

the string “ABCD” using two Holding Registers 40014 and 40015:

Character A = HEX 41, Character B = HEX 42 and HEX 4142 = Decimal 16706, so register 40014 = 16706

Character C = HEX 43, Character d = HEX 64 and HEX 4364 = Decimal 17252, so register 40015 = 17252

Appendix B:

Tag copying results between different data types

The following diagram maps tag copying results between different data types. Use this mapping to determine whether or not you can copy data between different types of tag:

Tag map results with different data types

		Destination Tag Type														
		BOOL	SINT	INT	UINT	DINT	UDINT	REAL	STRING	BOOL[8]	BOOL[16]	BOOL[32]	SINT[2]	SINT[4]	INT[2]	UINT[2]
Source Tag Type	BOOL							1								
	SINT	1		2,4	2,3	2,4	2,3	2		1	1	1			2,4	2,3
	INT	1	2		2,3	2,4	2,3	2		1	1	1	2	2		2,3
	UINT	1	2	2,4				2		1	1	1	2	2	2,4	
	DINT	1	2	2	2		2,3	2		1	1	1	2	2	2	2
	UDINT	1	2	2	2	2,4		2		1	1	1	2	2	2	2
	REAL	1	2	2	2	2,4	2,3			1	1	1	2	2	2	2
	STRING								2							
	BOOL[8]	1	1,4	1,4	1,3	1,4	1,3	1					1	1	1	1
	BOOL[16]	1	1	1,4	1,3	1,4	1,3	1		1			1	1	1	1
	BOOL[32]	1	1	1	1	1,4	1,3	1		1	1		1	1	1	1
	SINT[2]	1	2	2,4	2,3	2,4	2,3	2		1	1	1			2,4	2,3
	SINT[4]	1	2	2	2	2,4	2,3	2		1	1	1	2		2	2
	INT[2]	1	2	2	2	2,4	2,3	2		1	1	1	2	2		2,3
	UINT[2]	1	2	2	2	2,4	2,3	2		1	1	1	2	2	2,4	

Results Key

- Normal operation
- No data loss, results with notes
- Blocked by UI
- Target might not represent the source value
- Data Loss

CHART NOTES

1. Straight bit copy, length = min(SourceBitLength, DestBitLength)
2. Straight byte copy, length = min(SourceByteLength, DestByteLength)
3. Destination will be an unsigned value
4. Destination will be a signed value

REFERENCE

Type	Bit Length
BOOL	1
SINT	8
INT	16
UINT	16
DINT	32
UDINT	32
REAL	32
STRING	*

OPERATIONAL NOTES

- Destination tag is cleared to 0 (zero) before bits are copied
- Array order is [0] -> [0], [1] -> [1]
- Bit order is LSB to LSB. SINT[2] -> INT will place SINT[0] into bits 7-0 of the INT and SINT[1] into bits 15-8
- There is no endian swap being done at the high level. The protocol MAY do byte or word swap if required or if the user has selected to do so.

DATA LOSS EXAMPLES

SOURCE	DESTINATION
SINT :: -1 :: 0x81	INT :: 129 :: 0x0081
REAL :: 1234.123 :: 0xF0439A44	INT :: -6724 :: 0x9A44
REAL :: 1234.123 :: 0xF0439A44	UINT :: 39492 :: 0x9A44
DINT :: 8765432 :: 0x0085BFF8	INT :: 49144 :: 0xBFF8
INT[5] :: 1, 2, 3, 4, 5	INT[3] :: 1, 2, 3

Pay attention to type conversion with the SINT/USINT and DINT/UDINT data types. Both groups are the same size (number of bits), but the unsigned version will display data differently than expected if sent signed data. An example is sending -45 decimal to a USINT will result in 1101 0011 binary or 211 decimal. Data traveling the other direction will have the same kind of data change.

Appendix C:

Data type Value Ranges

Data type	length	Min Range	Max Range
REAL	32 bits (4 bytes)	-3.402823e+38	+3.402823e+38
INT	16 bits (2 bytes)	-32768	+32767
UINT (Unsigned INT)	16 bits (2 bytes)	0	65535
DINT (Double INT)	32 bits (4 bytes)	-2_147_483_648	+2_147_483_647
UDINT (Unsigned Double INT)	32 bits (4 bytes)	0	4294967295
SINT (Short INT)	8 bits (1 byte)	-128	+127
USINT (Unsigned Short INT)	8 bits (1 byte)	0	255

Table 2

REAL, DINT and UDINT data types require the **Word Swap**, SINT and USINT data types require the **Byte Swap**.

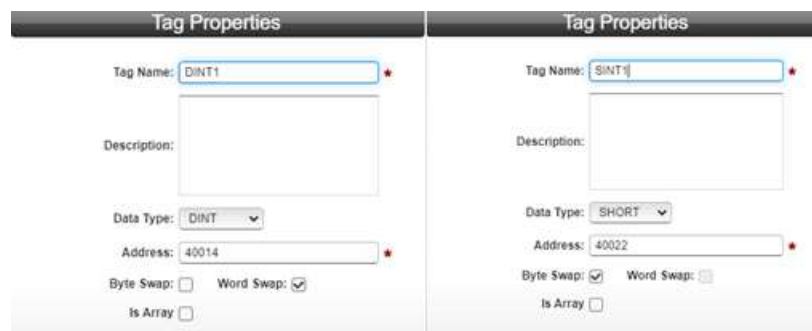


Figure C1

Something to note about the SINT and USINT data types is that **Modbus TCP always sends data in 16 bit chunks**, so when reading SINT and USINT data types (1 byte each). Since the offset is only one byte between Holding Registers 2 consecutive registers will be sent when only the first is requested. In other words, a SINT read is one WORD in size, even though there is only one byte of data. This can lead to some confusion unless the next Holding Register is left blank or the destination of the register can accept the 2 bytes that are delivered.

In Figure C2, the display format of the Live Tag Map Viewer is set to HEX and the value for 40022 is 12 hex. It shows up in the lower byte of the SINT1 register because of the Byte Swap. The value for 40023 is 34 hex. It shows up in the upper byte of the SINT1 register because of the Byte Swap. Only the lower byte gets delivered to the destination \$User.SINT1 in this case.

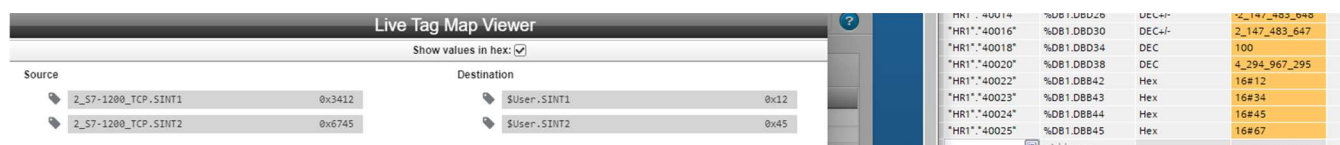


Figure C2

Both Holding Registers 40022 and 40023 are read even though the Source register is a SINT. If the Destination register was an INT data type it would be able to accept both bytes and be delivered intact.

Appendix D:

Troubleshooting Modbus TCP with The Trace Manager

The **Trace Manager** allows network traffic to be captured between the Gateway and a specific connected device for communications diagnostics purposes. To help diagnose communications issues with a Modbus TCP Device, define both the inbound and outbound devices to be that Device. That way both the queries from the Gateway, and the responses from the Device will be captured so complete transactions can be examined.

In Figure D1, the Modbus TCP Device name is 1_S71200_TCP and is selected as both the Inbound and Outbound devices. Also, the checkmark in the Capture Full Packet box is important so the packets are not truncated. If you only want to see TCP data, the checkmark can be left out of the Capture Full Packet box.

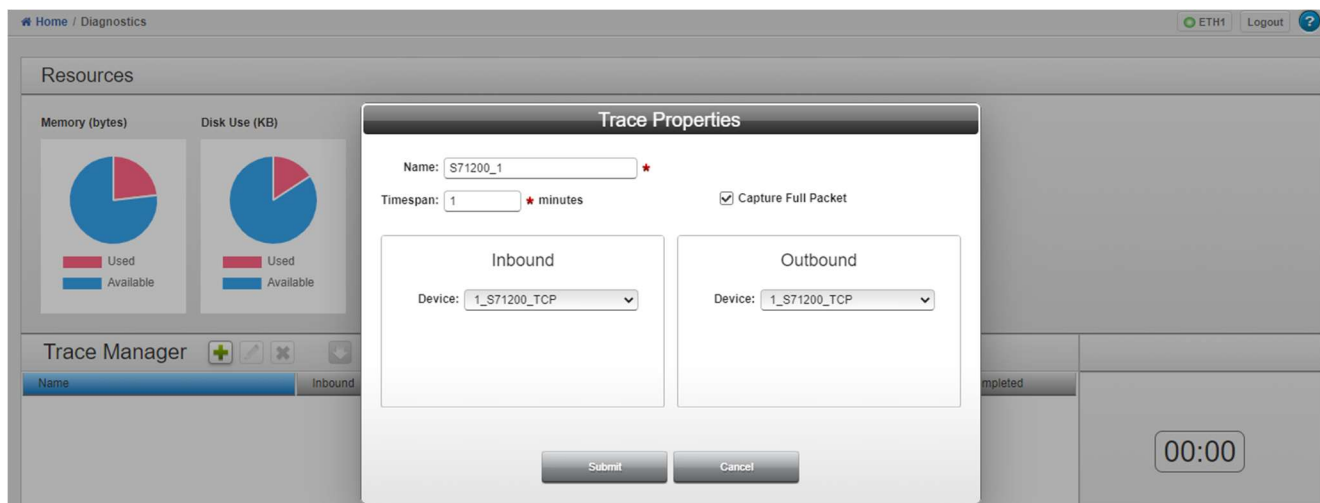


Figure D1

It's best to be a minimalist when performing communications troubleshooting. Turn off all Tag Maps and build a Tag Map that reads only a few tags in the target device to minimize the amount of data, while still allowing communications attempts. The Tag Map should update at a fairly slow pace – maybe once every 5 seconds, so the Trace data shows distinct periods of activity and quiet. To limit the activity even more, the destination tags should be \$User tags so the *only* activity is between the Gateway and Modbus TCP Device. No other devices are needed.

In Figure D2, the Tag Map Update Condition is set to 5 seconds, which allows distinct periods of quiet between attempts at communication. Also, the Destination registers are \$User tags which keeps the number of devices involved in the communications diagnostics to a minimum (quantity=2).

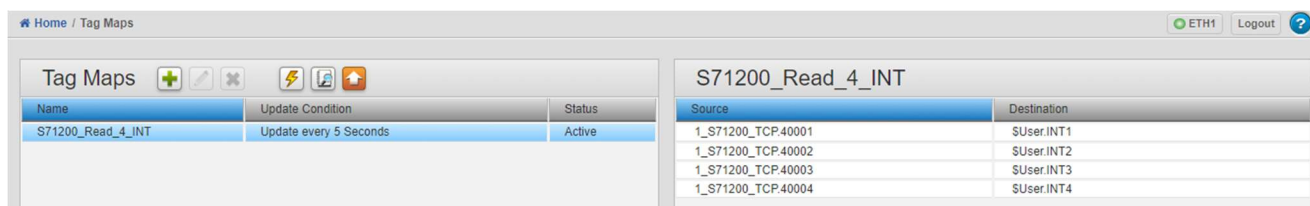


Figure D2

The diagnostics trace file that gets generated by the procedure can be opened by the WireShark program for examination. There will be an initial section, generated by the Test Device Connection button, then a period of quiet before the Tag Map is made active, then one or more Tag Map transactions will be captured.

Examining the capture file with WireShark takes a little understanding of network transactions, but here is a brief description of what the procedure will output on a system that is working normally.

The first transaction consists of 12 frames and the activity illustrates what happens when the Test Device Connection button is pressed while a Modbus TCP Device is selected. This transaction takes 0.009347 seconds to complete.

Figure D3 shows the cursor on Frame 4:

- This is a Modbus/TCP Query by the Gateway to the Modbus TCP Device with destination Port 502.
- This is a Function 3 transaction (Read Holding Register).
- The Reference Number is 0 (in the expanded part of the lower window)
- The Word Count is 1 (the quantity of registers to respond with).
- This Query means the Client is asking the Server for the data in the first Holding Register (40001).

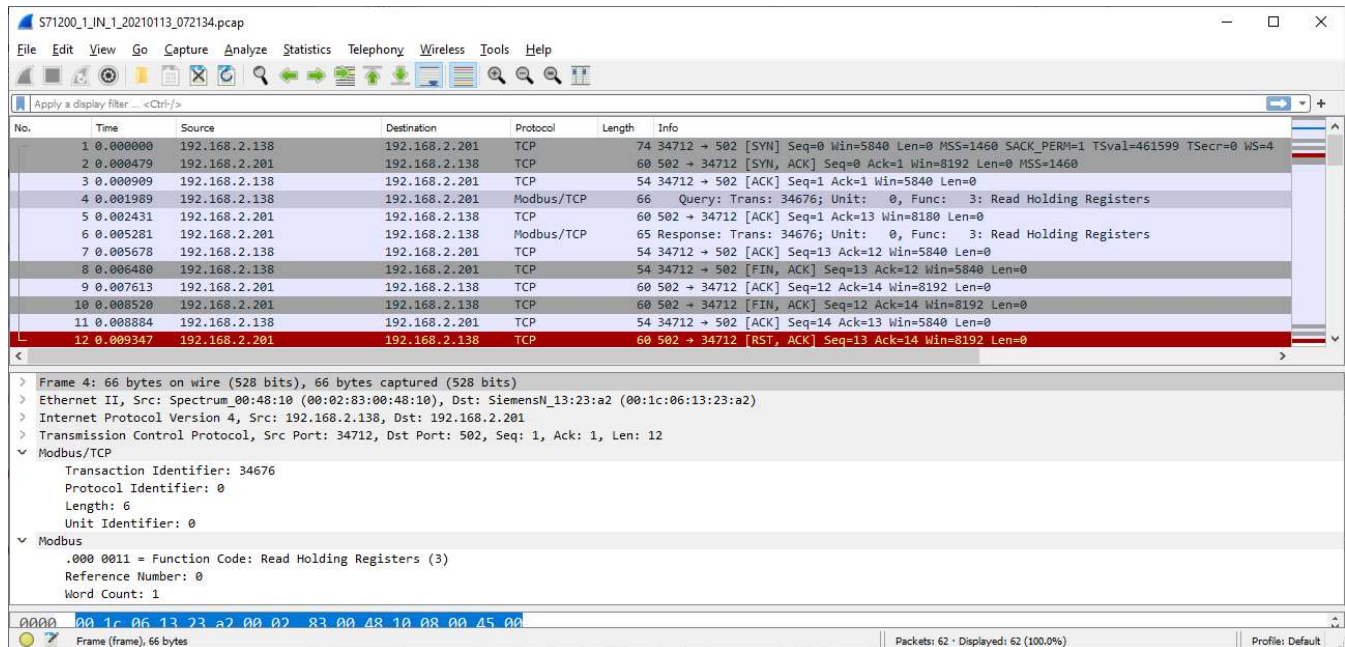


Figure D3

Figure D4 shows the cursor on Frame 6:

- This is a Response by the Modbus TCP Device to the Gateway from the source Port of 502.
- The byte count is 2 (one 16 bit Word).
- The Register Number is 0 and is a UINT16 data type.
- The Register 0 value is 12101.
- This Response means the Server is sending the Client the data (12101) in the first Holding Register (40001).

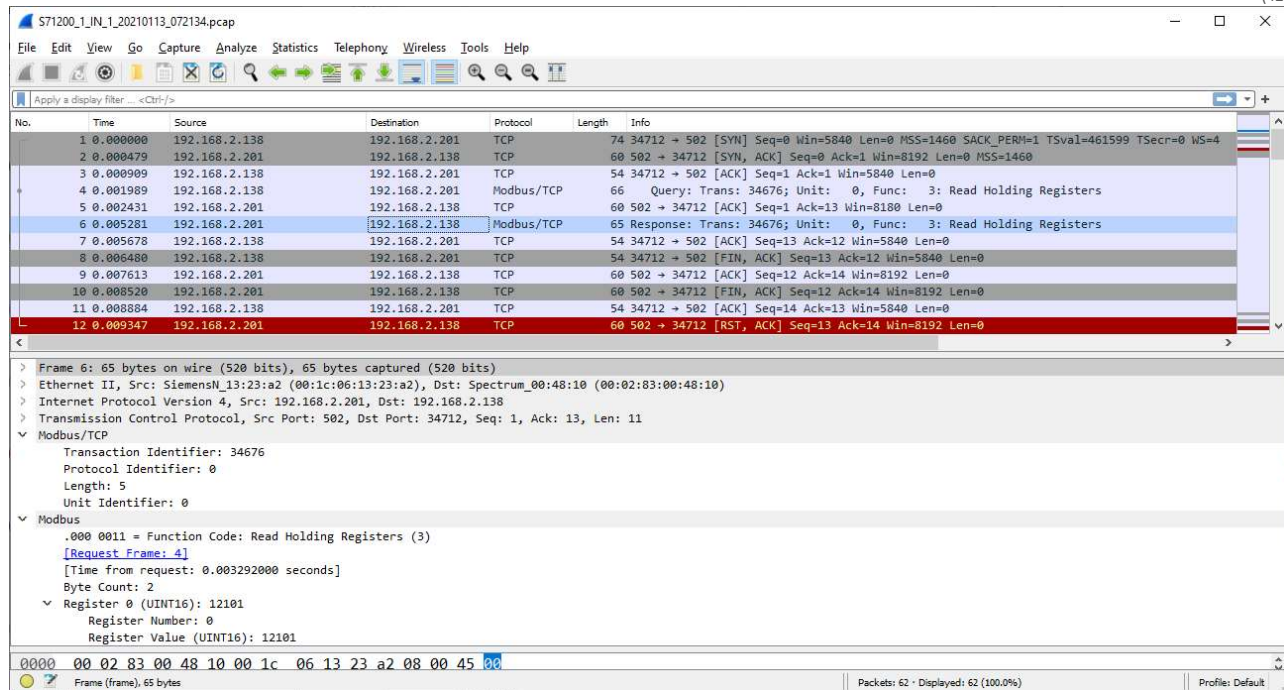


Figure D4

The next transaction consists of 17 frames (frames 13-29) and the activity illustrates what happens when the Gateway initiates a Tag Map asking a Modbus TCP Device to send the data in the first 4 Holding Registers (40001-40004). This transaction takes 0.057662 seconds to complete.

Here's the Query for the 4th Holding Register (Cursor is on Frame 26):

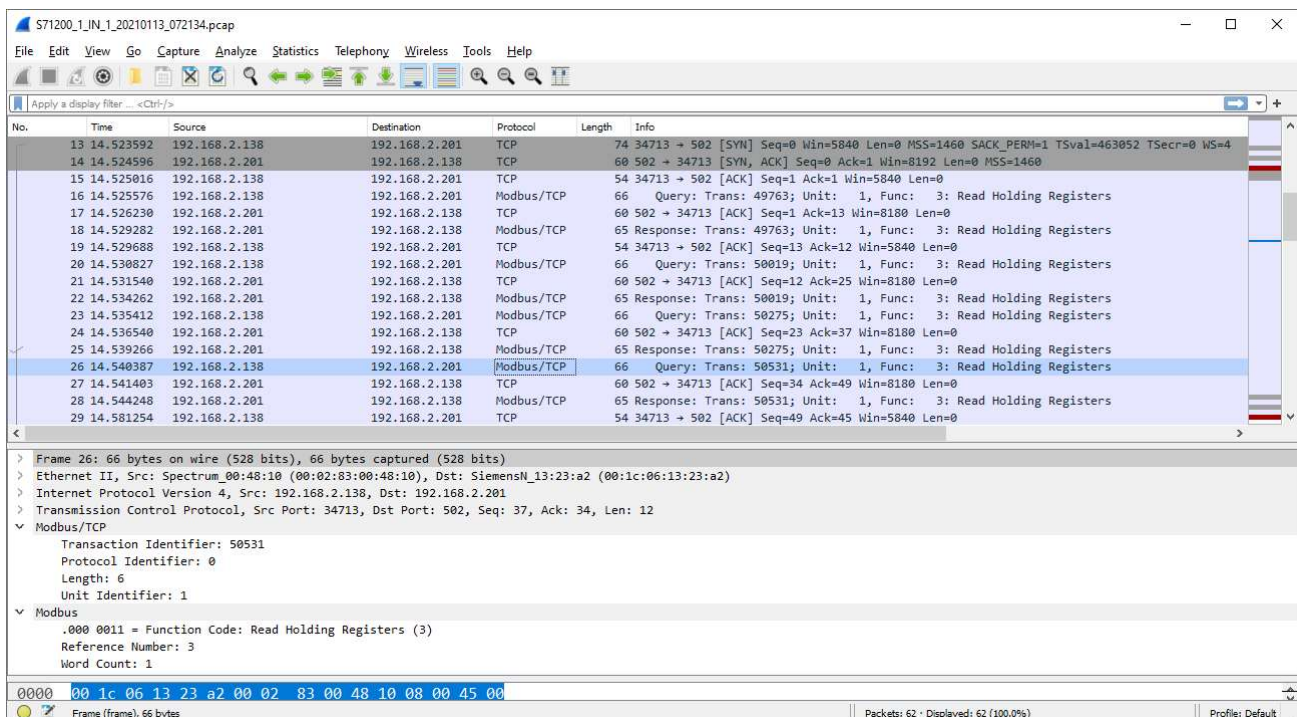


Figure D5

Here's the Response for the 4th Holding Register (Cursor is on Frame 28) with a returned value of 12404:

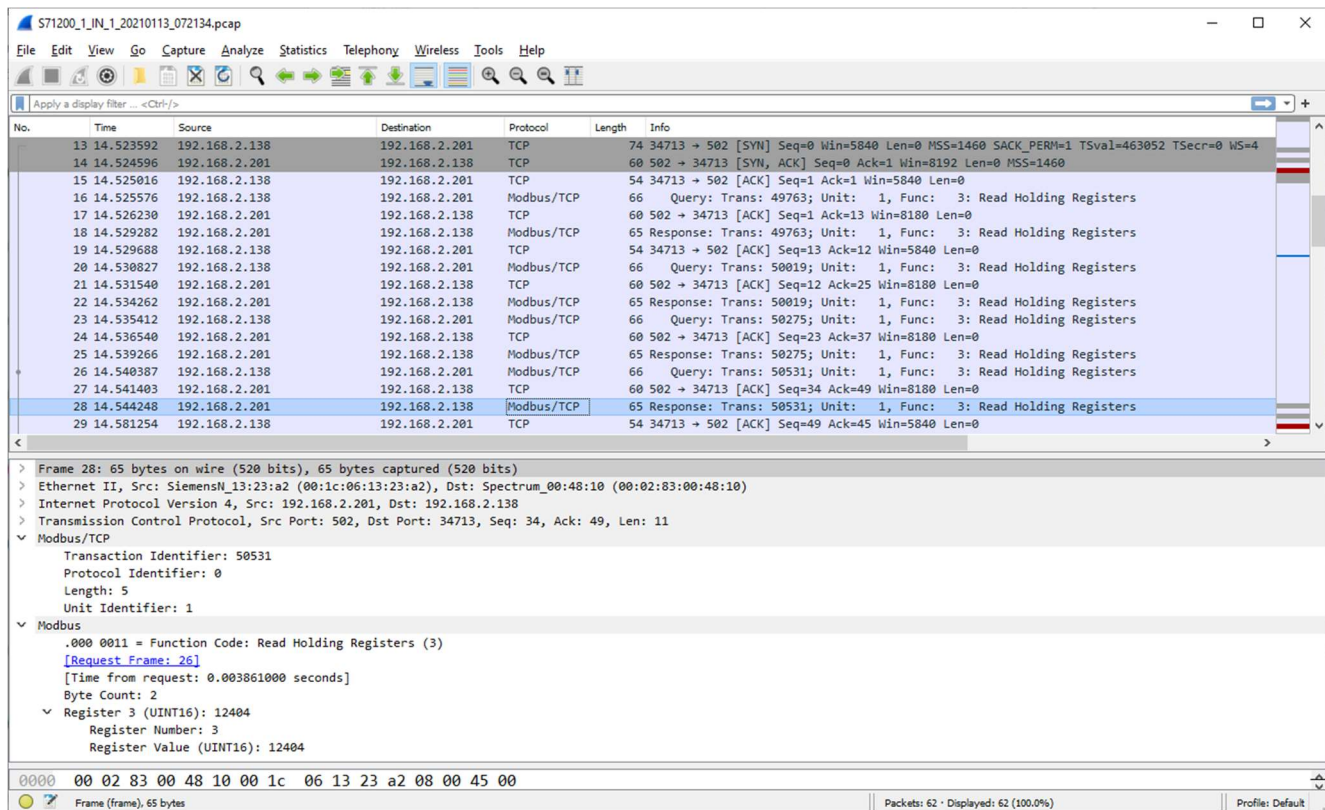


Figure D6

Procedure to generate a diagnostics trace file:

1. If not already present on your PC, download and install WireShark (WireShark.org).
2. Deactivate all Tag Maps using the ethernet port.
3. Make a simple tag map with a few registers from the Modbus Device as the Source, and \$User tags as the Destination. Make the Update Condition once every 5 seconds.
4. Navigate to the Diagnostics screen and add a Trace, pointing both the inbound and outbound to the Device of interest. Put a check in the Capture Full Packet box, if there is not one already.
5. Make a note of the time, then start the Trace.
6. Navigate to the Devices screen.
7. Select the Device of interest and press the Test Device Connection button.
8. Observe the contents of the resulting dialog box, then close it.
9. Navigate to the Tag Maps screen and activate the Tag Map with the Device of interest.
10. Wait about 10 seconds, maybe bring up the Live Tag Map Viewer for a few seconds and note what the condition of data flow is.
11. Deactivate the Tag Map.
12. Navigate to the Diagnostics screen and wait for the Trace to time out.
13. After the Trace times out, highlight it and press the Download button.
14. Navigate to the Logs screen, select the Engineering Log and press the Export Log button.
15. Find the 2 downloaded files and move them to an analysis directory for extraction from the zip format.
16. Extract the Trace file from the zip format, it will have a ".pcap" extension. Open it in WireShark for examination.