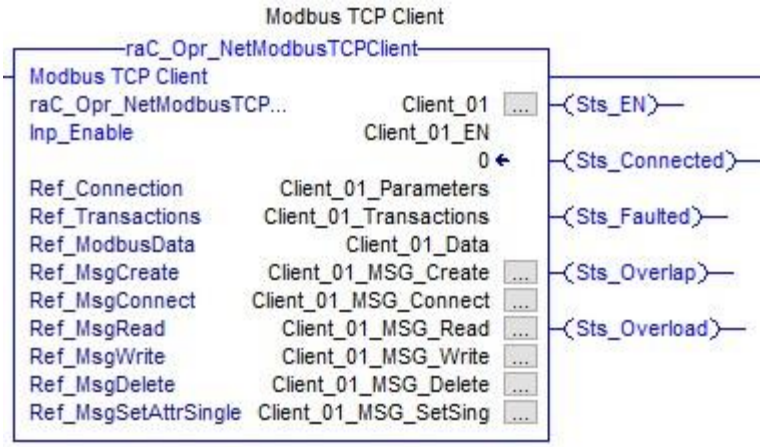
How to make Allen-Bradley CompactLogix and ControlLogix talk to EtherTRAK-2 and EtherTRAK-3 Modules using the RSLogix 5000 ModbusTCP Client Add On Instruction (AOI)

Introduction

This document describes the application and use of the Modbus TCP Client Add-On Instruction and how to make this instruction talk to an EtherTRAK-2 & 3 module with no protocol conversion taking place using only ModbusTCP.

First you will need to have the EtherTRAK module setup with an IP address and the unit number. Please see the technote on how to do this or the help file in Crimson 3.1 for this as this technote will not go into that directly.

Modbus TCP Client Add-On Instruction (AOI) allows users to implement Modbus TCP Client functionality into the Logix family of controllers. AOIs can be used standalone or can be added to an existing application following the directions outlined below.



Contents

Requirements

Hardware Requirements

The Modbus TCP Client code requires a ControlLogix or CompactLogix controller with an EtherNet/IP module that supports Logix Sockets functionality. See Knowledgebase technote 470690 for complete list of controllers and modules. https://rockwellautomation.custhelp.com/app/answers/detail/a_id/470690

Software Requirements

The Modbus TCP Client AOI code supports Logix controller revisions 20 and higher.

Memory Requirements

First instance of the Modbus TCP Client AOI uses about 108 Kbytes of memory.
Each additional AOI instance requires about 22.5 Kbytes of memory.
These estimates based on the ControlLogix 5570 family of controllers.
Please note that some Compactlogix controllers have a starting memory size as low as 384Kbytes. This code can take a significant amount of memory in smaller CompactLogix controllers.

Functional Requirements and Description

Supported Modbus Function Codes

Bit Level Commands

Function Code	Name	Description	Supported Values	Modbus Range
01	Read Coils	This function code is used to read contiguous status of coils in a remote device (0xxxx addresses). The coils in the response message are packed as one coil per bit of the data field.	Local Address: 0 to 1023 Server Address: 0 to 65535 Length: 1 to 256 coils	Local Address 00001-01024 Server Address 000001-065536
02	Read Discrete Inputs	This function code is used to read contiguous status of discrete inputs in a remote device (1xxxx addresses). The inputs in the response message are packed as one coil per bit of the data field.	Local Address: 0 to 1023 Server Address: 0 to 65535 Length: 1 to 256 Inputs	Local Address 10001-11024 Server Address 10001-165536
05	Write Single Coil	This function code is used to write a single coil to either ON or OFF in a remote device (0xxxx addresses).	Local Address: 0 to 1023 Server Address: 0 to 65535	Local Address 00001-01024 Server Address 000001-065536
15	Write Multiple Coils	This function code is used to write to one or more coils in a sequence of coils to either ON or OFF in a remote device (0xxxx addresses).	Local Address: 0 to 1023 Server Address: 0 to 65535 Length: 1 to 256 coils	Local Address 00001-01024 Server Address 000001-065536

Word Level Commands

Function Code	Name	Description	Supported Values	Modbus Range
03	Read Holding Registers	This function code is used to read the contents of a contiguous block of holding registers (4xxxx addresses) in a remote device.	Local Address: 0 to 1023 Server Address: 0 to 65535 Length: 1 to 120 registers	Local Address 40001-41024 Server Address 400001-465536

04	Read Input Registers	This function code is used to read the contents of a contiguous block of input registers (3xxxx addresses) in a remote device.	Local Address: 0 to 1023 Server Address: 0 to 65535 Length: 1 to 120 input registers	Local Address 30001-31024 Server Address 300001-365536
06	Write a Single Holding Register	This function code is used to write to single holding register (4xxxx addresses) in a remote device	Local Address: 0 to 1023 Server Address: 0 to 65535	Local Address 40001-41024 Server Address 400001-465536
16	Write Multiple Holding Registers	This function code is used to write to contiguous holding registers (4xxxx addresses) in a remote device.	Local Address: 0 to 1023 Server Address: 0 to 65535 Length: 1 to 120 registers	Local Address 40001-41024 Server Address 400001-465536

Data Format

The Modbus TCP Client AOI supports Modbus TCP standard format of big-endian protocol. This means that the most significant byte of a 16-bit value is sent before the least significant byte.

Implementation

Modbus TCP Client AOI implementation

Using Periodic Task

It's recommended to add AOIs into a Periodic task with Rate of 10ms (or higher).
Slower rates will reduce controller load and reduce performance.
Faster task rates will increase performance but will add a significant load to the controller.

See Performance Data section for details.

Rung Import and tag naming changes

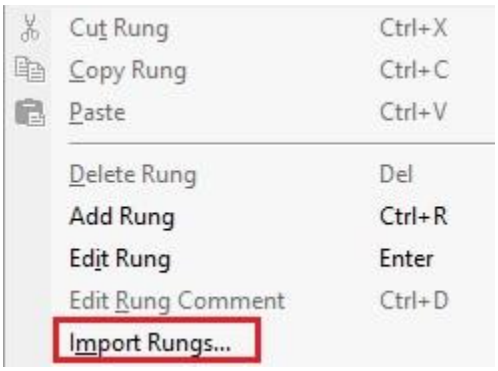
The pre-configured Add-On Instructions are supplied in a Rung format.

The Rung Import format must be used to implement the AOI.

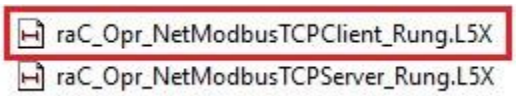
***Important**:*
Use only the Rung Import process.
Do not use Copy/Paste functionality or add these AOIs using Instructions tool bar. Doing this will remove configurations from pre-configured message instructions, making AOIs non-functional.

Rung Import process for Modbus TCP Client AOI

1. Open a Ladder Routine within your application
2. Right click on any empty area and select ***Import Rungs***

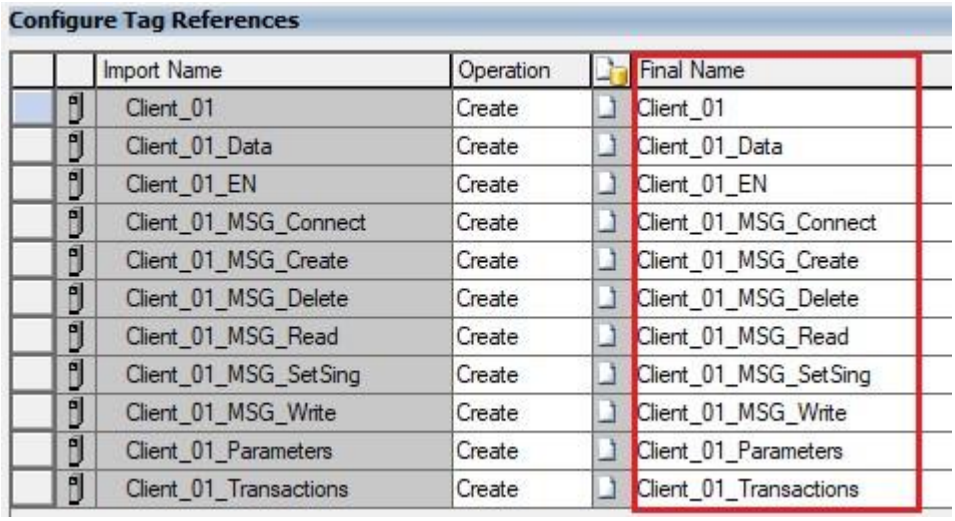


3. Select ***raC_Opr_NetModbusTCPClient_Rung.L5X*** file and click ***Import***



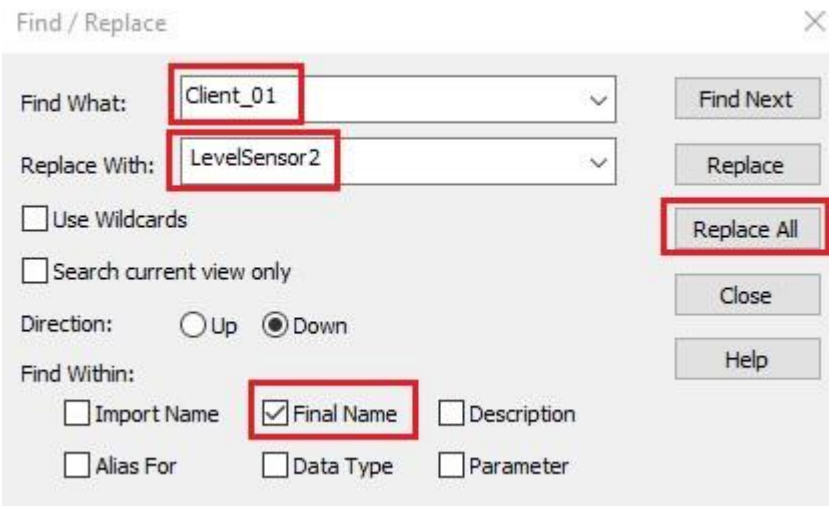
4. When Import Configuration Dialog opens, select ***Tags***

5. You can leave final names as-is or change them to accommodate your application.



6. To change Final Names click ***Find/Replace*** button

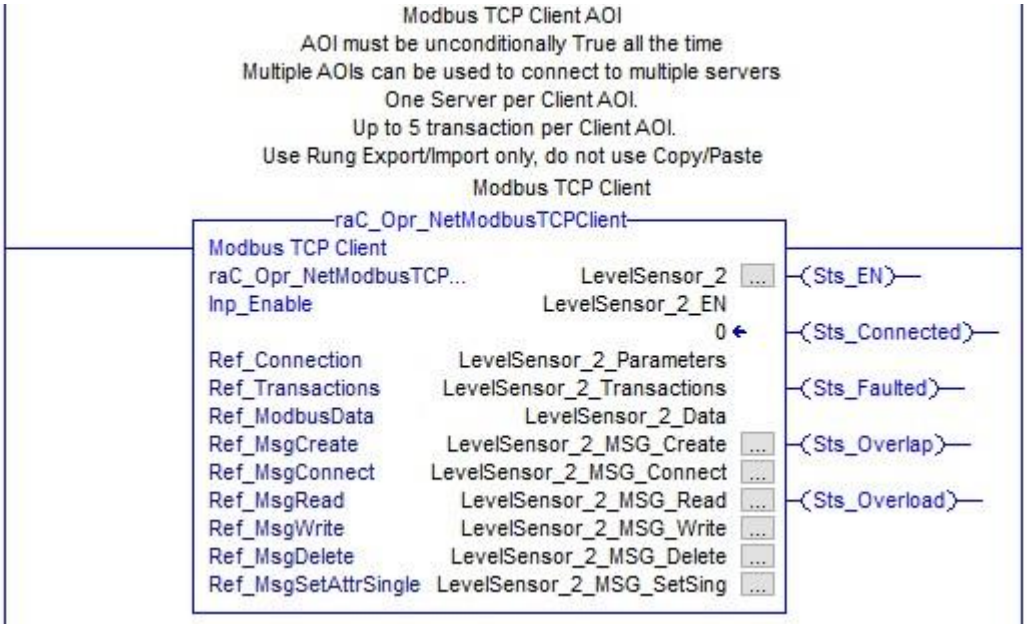
When Dialog opens, replace default name **Client_01** with desired prefix, verify that Final Names Box is checked then click ***Replace All***



Close Find/Replace dialog and verify **Final Names**

Final Name
LevelSensor2
LevelSensor2_Data
LevelSensor2_EN
LevelSensor2_MSG_Connect
LevelSensor2_MSG_Create
LevelSensor2_MSG_Delete
LevelSensor2_MSG_Read
LevelSensor2_MSG_SetSing
LevelSensor2_MSG_Write
LevelSensor2_Parameters
LevelSensor2_Transactions

7. Click ***Ok*** to finish the Import process
   The new rung should look like shown below without any errors



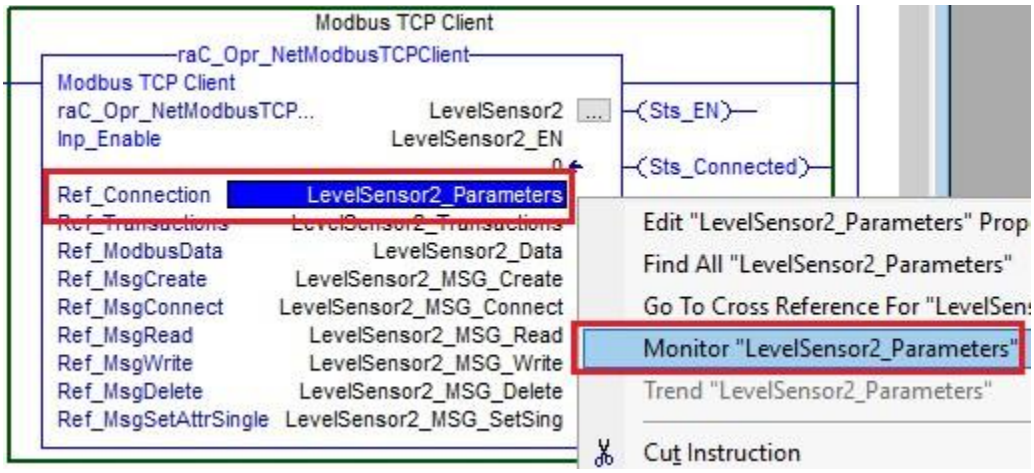
8. Repeat Steps 2-7 if application requires additional Clients. DO NOT Copy/Paste.

Configure Local Operational Parameters

Modbus TCP Client requires a local EtherNet/IP module that supports Logix Sockets. See Requirements section for details.

In this section we will link Modbus TCP Client AOI to this EtherNet/IP module.

1. Right Click on the tag attached to the **Ref_Connection** parameter and select ***Monitor "…"***



2. Expand Parameters tag. Specify the slot of the Local EtherNet/IP module.

LevelSensor2_Parameters	raC_UDT_Modbus...	{...}
LevelSensor2_Parameters.LocalSlot	SINT	0
LevelSensor2_Parameters.LocalAddress	STR0016	''
LevelSensor2_Parameters.DestAddress	STR0016	''
LevelSensor2_Parameters.DestinationPort	DINT	502

For 1756 ControlLogix processors specify the actual slot of desired 1756-EN2T(R) module.
For 1756-L8xE controllers using the built in Ethernet port specify the 1756-L8xE controller slot.
For CompactLogix 5370, 5380, 5480 controllers leave **.LocalSlot** at 0.

3. Specify the **.LocalAddress** of the EtherNet/IP module.

LevelSensor2_Parameters	raC_UDT_Modbus...	{...}
LevelSensor2_Parameters.LocalSlot	SINT	0
LevelSensor2_Parameters.LocalAddress	STR0016	''
LevelSensor2_Parameters.DestAddress	STR0016	''
LevelSensor2_Parameters.DestinationPort	DINT	502

For CompactLogix 5380 and 5480 in **Dual IP Mode only**, specify the IP Address of the Local Ethernet connection used for Modbus TCP communications. **Leave this field blank for all other cases.**

4. Specify Ethernet IP Address of the Modbus Server device. This address must be specified and cannot remain blank.

LevelSensor2_Parameters	raC_UDT_Modbus...	{...}
LevelSensor2_Parameters.LocalSlot	SINT	0
LevelSensor2_Parameters.LocalAddress	STR0016	''
LevelSensor2_Parameters.DestAddress	STR0016	'192.168.1.150'
LevelSensor2_Parameters.DestinationPort	DINT	502

5. Leave Default Modbus TCP port at 502. This value is Modbus TCP protocol standard.
6. Start Modbus TCP Client by setting tag attached to **Inp_Enable** parameter to 1.

If you change any of these Parameters during operation be sure to reset and then set the AOI **Inp_Enable** parameter tag.

Configure Data Transactions

1. Expand Transactions tags to expose Transaction 0 member tags

LevelSensor2_Transactions	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0]	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0].Enabled	BOOL	0
LevelSensor2_Transactions[0].PollInterval	DINT	1000
LevelSensor2_Transactions[0].TransType	INT	3
LevelSensor2_Transactions[0].StationID	SINT	0
LevelSensor2_Transactions[0].BeginAddress	DINT	1
LevelSensor2_Transactions[0].Count	INT	100
LevelSensor2_Transactions[0].LocalAddress	INT	6
LevelSensor2_Transactions[0].TransComplete	BOOL	1
LevelSensor2_Transactions[0].TransStat	INT	0
LevelSensor2_Transactions[0].Diagnostic	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[1]	raC_UDT_ModbusCl...	{...}

2. Set Polling Interval value in milliseconds.
   Default value is 1000 (1 second).
   Minimum value is 80 msec. Any transaction with polling rate below 80ms will be polled at 1 second rate.

LevelSensor2_Transactions	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0]	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0].Enabled	BOOL	0
LevelSensor2_Transactions[0].PollInterval	DINT	1000
LevelSensor2_Transactions[0].TransType	INT	3
LevelSensor2_Transactions[0].StationID	SINT	0
LevelSensor2_Transactions[0].BeginAddress	DINT	1
LevelSensor2_Transactions[0].Count	INT	100
LevelSensor2_Transactions[0].LocalAddress	INT	6
LevelSensor2_Transactions[0].TransComplete	BOOL	1
LevelSensor2_Transactions[0].TransStat	INT	0
LevelSensor2_Transactions[0].Diagnostic	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[1]	raC_UDT_ModbusCl...	{...}

3. Set Modbus Function code in **TransType** tag. See Supported Modbus Function Codes section for the list of supported commands.

LevelSensor2_Transactions	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0]	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0].Enabled	BOOL	0
LevelSensor2_Transactions[0].PollInterval	DINT	1000
LevelSensor2_Transactions[0].TransType	INT	3
LevelSensor2_Transactions[0].StationID	SINT	0
LevelSensor2_Transactions[0].BeginAddress	DINT	1
LevelSensor2_Transactions[0].Count	INT	100
LevelSensor2_Transactions[0].LocalAddress	INT	6
LevelSensor2_Transactions[0].TransComplete	BOOL	1
LevelSensor2_Transactions[0].TransStat	INT	0
LevelSensor2_Transactions[0].Diagnostic	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[1]	raC_UDT_ModbusCl...	{...}

4. Set StationID (UID in previous versions) value **only if required by the peer Server device**. In most of cases this field is ignored by the Modbus TCP Server and can remain at 0.
   Refer to the server documentation for details. Values 0-127 can be entered directly.

   Values 128-255 should be converted to a Hexadecimal number and entered in Hex format (16#xx). By default, these values will be displayed as negative number under StationID field. Change display Style to Hex to see the hexadecimal value.

LevelSensor2_Transactions	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0]	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0].Enabled	BOOL	0
LevelSensor2_Transactions[0].PollInterval	DINT	1000
LevelSensor2_Transactions[0].TransType	INT	3
LevelSensor2_Transactions[0].StationID	SINT	0
LevelSensor2_Transactions[0].BeginAddress	DINT	1
LevelSensor2_Transactions[0].Count	INT	100
LevelSensor2_Transactions[0].LocalAddress	INT	6
LevelSensor2_Transactions[0].TransComplete	BOOL	1
LevelSensor2_Transactions[0].TransStat	INT	0
LevelSensor2_Transactions[0].Diagnostic	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[1]	raC_UDT_ModbusCl...	{...}

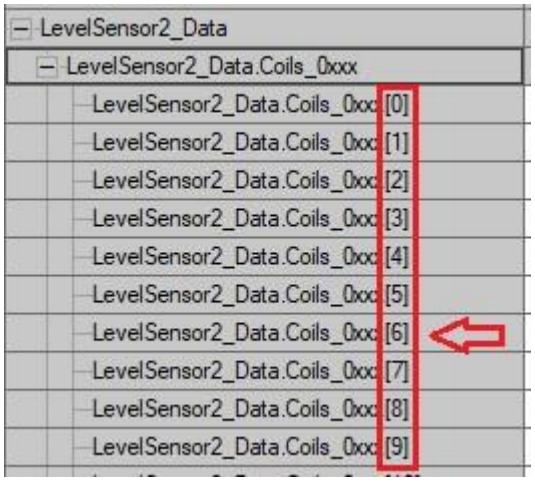
5. Set **BeginAddress** tag. The value represents the beginning address in remote device (Modbus TCP Server) to read from or write to.
   Depending on the function used above, values 0…65535 represent Modbus addresses 000001065536, 100001-165536, 300001-365536, 400001-465536 respectively.
   AOI version prior to Ver 2.2.0 had these values limited to 9999

LevelSensor2_Transactions	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0]	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0].Enabled	BOOL	0
LevelSensor2_Transactions[0].PollInterval	DINT	1000
LevelSensor2_Transactions[0].TransType	INT	3
LevelSensor2_Transactions[0].StationID	SINT	0
LevelSensor2_Transactions[0].BeginAddress	DINT	1
LevelSensor2_Transactions[0].Count	INT	100
LevelSensor2_Transactions[0].LocalAddress	INT	6
LevelSensor2_Transactions[0].TransComplete	BOOL	1
LevelSensor2_Transactions[0].TransStat	INT	0
LevelSensor2_Transactions[0].Diagnostic	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[1]	raC_UDT_ModbusCl...	{...}

6. Set **Count** tag. The value represents the number of items in remote device (Modbus TCP Server) to read from or write to.

LevelSensor2_Transactions	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0]	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0].Enabled	BOOL	0
LevelSensor2_Transactions[0].PollInterval	DINT	1000
LevelSensor2_Transactions[0].TransType	INT	3
LevelSensor2_Transactions[0].StationID	SINT	0
LevelSensor2_Transactions[0].BeginAddress	DINT	1
LevelSensor2_Transactions[0].Count	INT	100
LevelSensor2_Transactions[0].LocalAddress	INT	6
LevelSensor2_Transactions[0].TransComplete	BOOL	1
LevelSensor2_Transactions[0].TransStat	INT	0
LevelSensor2_Transactions[0].Diagnostic	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[1]	raC_UDT_ModbusCl...	{...}

7. Set **LocalAddress** tag. The value represents the beginning address in this programs' "_Data" arrays.



LevelSensor2_Transactions	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0]	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0].Enabled	BOOL	0
LevelSensor2_Transactions[0].PollInterval	DINT	1000
LevelSensor2_Transactions[0].TransType	INT	3
LevelSensor2_Transactions[0].StationID	SINT	0
LevelSensor2_Transactions[0].BeginAddress	DINT	1
LevelSensor2_Transactions[0].Count	INT	100
LevelSensor2_Transactions[0].LocalAddress	INT	6
LevelSensor2_Transactions[0].TransComplete	BOOL	1
LevelSensor2_Transactions[0].TransStat	INT	0

8. Start Transaction by setting the .**Enabled** tag to 1.

LevelSensor2_Transactions	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0]	raC_UDT_ModbusCl...	{...}
LevelSensor2_Transactions[0].Enabled	BOOL	1
LevelSensor2_Transactions[0].PollInterval	DINT	1000
LevelSensor2_Transactions[0].TransType	INT	3
LevelSensor2_Transactions[0].StationID	SINT	0

Implementation Restrictions

1. Implementation must be done using Import Rung function only to preserve Message instruction configurations. Do not use Copy/Paste as it will not bring complete Message instruction configurations and tags. Do not use Search/Replace tags once rung is implemented. All replacement can be done only during rung import.

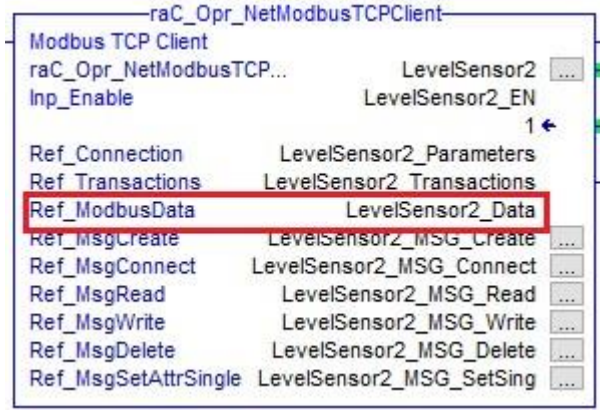
2. Multiple instances of Client AOI are supported per controller.
   Each instance must use own set of backing and Message tags, however **"…_Data"** tag structure can be shared between AOIs.

3. Modbus TCP Server and Modbus TCP Client AOIs can reside in the same program. However Server applications may cause a temporary Client disconnection due to the shared Logix Sockets object.

4. When implemented in ControlLogix Redundancy system, user should expect at least a 5 second loss of Modbus communications after a controller switchover attributed to the Modbus TCP Client AOI. There can be additional Modbus communications delays attributed to the behavior of the exact Server device being used.

Monitoring Modbus TCP Client operations

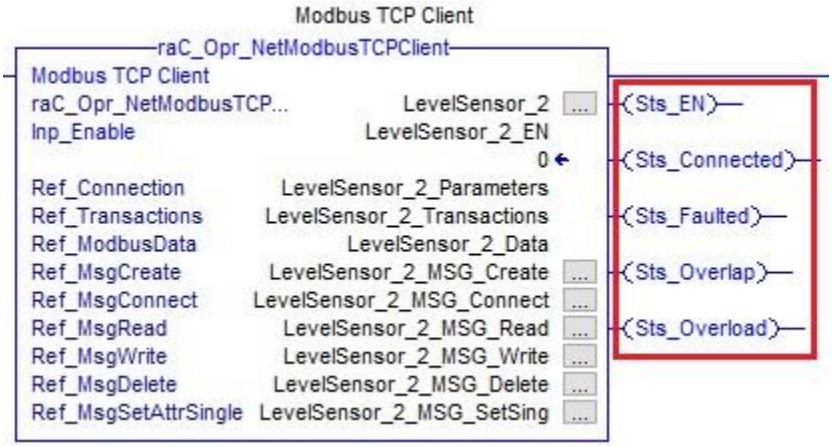
1. Modbus Tags are located under **Ref_ModbusData** parameter tag.



This tag contains four separate data areas for coils (0xxxx), discrete inputs (1xxxx), input registers (3xxxx) and holding registers (4xxxx). These tag values can be read and populated by the user application without any restrictions.

LevelSensor_2_Data	raC_UDT_Mod...
LevelSensor_2_Data.Coils_0xxx	BOOL[1024]
LevelSensor_2_Data.DiscInputs_1xxx	BOOL[1024]
LevelSensor_2_Data.InpRegisters_3xxx	INT[1024]
LevelSensor_2_Data.HoldRegisters_4xxx	INT[1024]

2. Modbus TCP AOI Status Bits



a. **Sts_EN** output indicates that Modbus TCP Client functionality is enabled.
b. **Sts_Connected** output indicates that Client connection request was accepted by the Server. It does not indicate the active data flow. The status of individual transactions should be checked to verify the data exchange.
c. **Sts_Faulted** output indicates that one of the message instructions is faulted.

d. **Sts_Overlap** output indicates that one or more transaction is not completing before next trigger
e. **Sts_Overload** output indicates excessive Overlaps in one or more transactions.

3. Individual Transaction status information. These tags exist for each of the 5 built in transactions. Transaction 0-4).

   TransComplete bit is set when requested transaction was completed. It's cleared by the program when next transaction is requested.
   TransStatus value indicates the current status of transaction.
   0 = success, 1 = in process, 2 = retry, -1 = exception

LevelSensor2_Transactions[0].TransComplete	BOOL	1
+ LevelSensor2_Transactions[0].TransStat	INT	0

4. Transaction Diagnostic data structure provides internal dynamic information while the transaction is active. **Do not write to these tags.**

− LevelSensor_2_Transactions[0].Diagnostic	raC_UDT_Mod...	{...}
+ LevelSensor_2_Transactions[0].Diagnostic.Request	STR0462	'$0F$A8$00$00$0...
LevelSensor_2_Transactions[0].Diagnostic.ReqBuilt	BOOL	0
LevelSensor_2_Transactions[0].Diagnostic.Overlap	BOOL	0
LevelSensor_2_Transactions[0].Diagnostic.Overload	BOOL	0
+ LevelSensor_2_Transactions[0].Diagnostic.TransID	INT	4008
+ LevelSensor_2_Transactions[0].Diagnostic.TransLastError	INT	0
+ LevelSensor_2_Transactions[0].Diagnostic.IntervalTimer	TIMER	{...}
+ LevelSensor_2_Transactions[0].Diagnostic.NoReplyCounter	COUNTER	{...}
+ LevelSensor_2_Transactions[0].Diagnostic.PendingTimer	TIMER	{...}

Performance data

Modbus Client performance can be affected by many factors including: periodic task rate, performance of the Server device, speed of Client controller, how busy the Client controller is, network performance, network card, number of Clients in the controller, the number of active transactions etc.

Below are typical performance expectations when using an L7x Controller with a **10ms** periodic task:

Number of Transaction Enabled	**Recommended Minimum *PollInterval* for all Transactions**
1	80 mS
2	130 mS
3	220 mS
4	300 mS

5	380 mS

When using the recommended **PollInterval** settings or greater, you can expect the actual data delivery to be very close to the **PollInterval**.

NOTE:
If selected **PollInterval** settings are marginally too fast you will likely see occasional **Sts_Overlap** errors and your system should work reasonably well.
If selected **PollInterval** settings are extremely out of line, you will see both **Sts_Overlap** and **Sts_Overload** errors and your system will not work reliably. If you are getting **Sts_Overload** errors, you must adjust your **PollInterval** settings.

Error Codes

Connection specific codes

The following local codes stored in the **Sts_LastError** AOI parameter

0 No fault

-2 Socket Write Message error

-4 Socket Read Message error

-8 Create Socket Message failed

To troubleshoot Socket faults above, review the corresponding message fault code and see publication **EtherNet/IP Socket Interface** for error code details.

-12 AOI Enable signal was removed with connection active

Transaction specific Codes

The following codes stored under the **Ref_Transaction** parameter separately for each transaction under the ***<TransactionTagName>*[x].Diagnostic.TransLastError** parameter tag.

-12 Unsupported local transaction code specified under the **TransType** tag value. See pages 4-5 for the list of supported codes.

-13 Illegal Local address range specified under **LocalAddress** and Count tags. See pages 4-5 for the list of supported local addresses.

Modbus TCP exceptions

Modbus TCP Servers may respond to this Client AOI with the exception codes.
These are stored under the **Ref_Transaction** parameter separately for each transaction under the ***<TransactionTagName>*[x].Diagnostic.TransLastError** parameter tag.

An exception is indicated by the 8th bit being sent in the return transaction type, for example transaction type of 3 would be returned as 16#83 (hex) or 131 (decimal). The next byte will contain Modbus Exception code. These Exception codes are generated by the peer server and may deviate from the standard Modbus Exception list.

Here is the partial list of the most common Modbus Exception Codes:

01 Illegal Function. The function code received in the query is not an allowable action for the server.

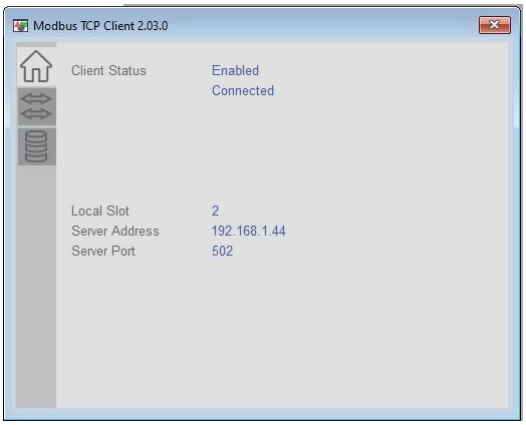
02 Illegal Data Address. The data address received in the query is not an allowable address for the server or the combination of starting address and length is invalid.

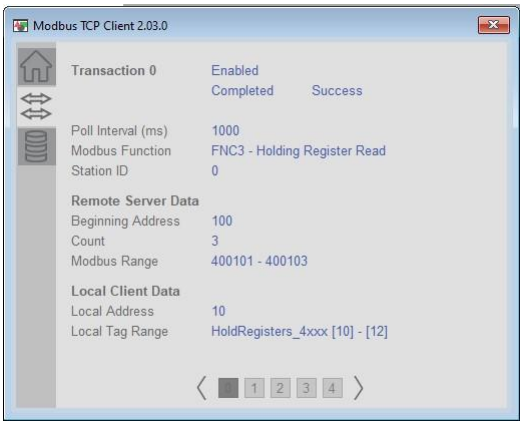
03 Illegal Data or Length request. A value contained in the query data or length field is not an allowable value for the server.

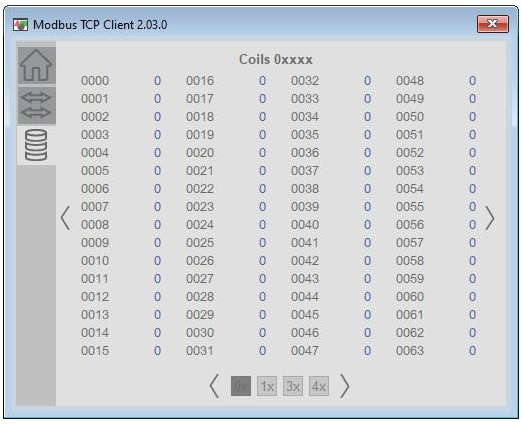
Visualization

Starting with version 2.03.00, the optional FactoryTalk View SE, ME and View Designer HMI displays provided to simplify implementation and troubleshooting.

These displays allow users to see Modbus TCP Client status, settings of the individual transactions and Modbus Data arrays.







Installation instructions for HMI Displays provided separately.

The supporting AOI code is located at the bottom of the AOI ladder logic and can be removed (along with HMI tags) to reduce AOI memory footprint.

Revision History

The latest version 2.03.00 is recommended for use in all new applications.

1. Revision 1.02 – Initial Release in Ladder Program format. If you are currently using this code in an existing application, you may continue to do so.

2. Client Revision 2.00.00 – Initial Release in Add-On Instruction format.

    2.1. Enhancements
        2.1.1. Re-written code in Add-On instruction format
        2.1.2. Reduced memory requirements
        2.1.3. Simplified implementation and configuration

    2.2. Corrected Anomalies
        2.2.1. None

    2.3. Known Anomalies
        2.3.1. None

3. Client Revision 2.00.01 - Update
    3.1. Enhancements
        3.1.1. None

    3.2. Corrected Anomalies
        3.2.1. Minor logic correction related to the local IP Address

    3.3. Known Anomalies
        3.3.1. None

4. Client Revision 2.01.00 – Update May 2020
   4.1. Enhancements
      4.1.1. Corrected Overload and Overlap indicators in Fault and not Enabled modes. This correction does not change indicator functionality under normal conditions.

   4.2. Corrected Anomalies
      4.2.1. None

   4.3. Known Anomalies
      4.3.1. None

5. Client Revision 2.02.00 – Update July 2020
   5.1. Enhancements
      5.1.1. Extended Remote address range to 0-65535.
         BeginAddress transaction member datatype changed to DINT and boundary check is introduced.
      5.1.2. Forced ***LocalSlot*** parameter to 0 for CompactLogix controllers
      5.1.3. Improved initialization code.

   5.2. Corrected Anomalies
      5.2.1. None

   5.3. Known Anomalies
         None

6. Client Revision 2.03.00 – Update April 2021
   6.1. Enhancements
      6.1.1. Replaced Transaction Station Address tag name **UID** with **StationID**.
      6.1.2. Added Read-Only visualization interface tags.

   6.2. Corrected Anomalies
      6.2.1. Added Count boundary size check before sending the request.

   6.3. Known Anomalies
      6.3.1. None

7. Client Revision 2.04.00 – Update September 2022
   7.1. Enhancements
      7.1.1. Improved incoming malformed packet handling.
      7.1.2. Added Error codes to this manual.

7.2. Corrected Anomalies

7.2.1. Resolved occasional minor faults Type 4 Code 56 and Type 4 Code 51

7.2.2. Corrected Overload logic for transactions higher than 0

7.3. Known Anomalies 7.3.1. None.