AN-023

Protege GX Modbus Server Integration

Application Note

Last Published: 18-Apr-23 01:37 PM

Contents

Introduction

Protege GX is an enterprise level integrated access control, intrusion detection and building automation solution with a feature set that is easy to operate, simple to integrate and effortless to extend.

Communication is over a proprietary high speed protocol across an encrypted local area network and AES encrypted proprietary RS-485 module network. Using modular-based hardware design, system installers have the flexibility to accommodate any installation, small or large, residential or commercial.

The Modbus protocol was developed by Modicon for industrial automation applications and provides a commonly available means of connecting industrial electronic instruments. Modbus is relatively easy to deploy and maintain and remains the most widely available protocol for connecting industrial devices.

Protege GX offers two methods of integrating with Modbus systems:

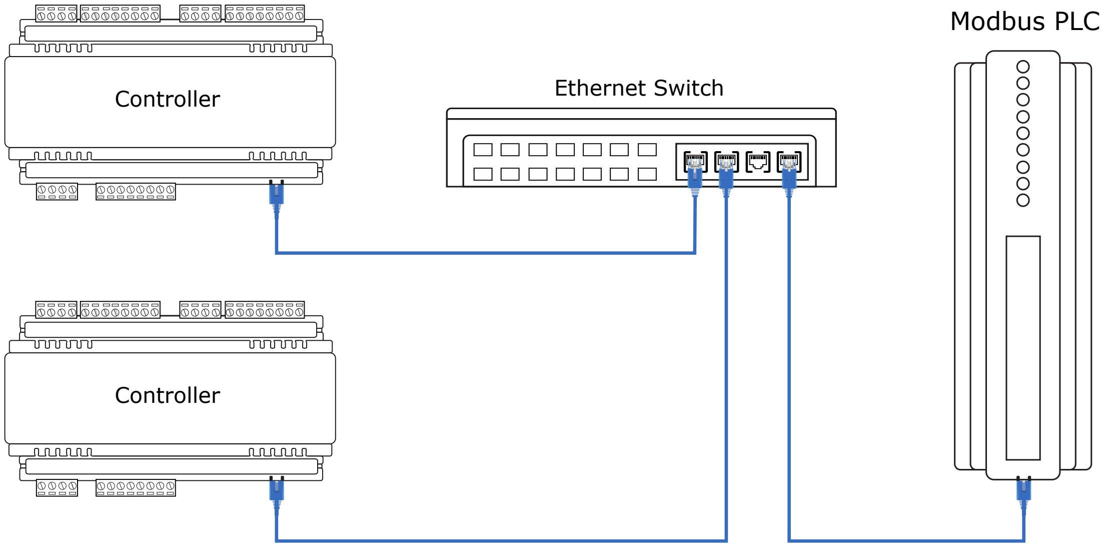
Modbus Server Integration	Modbus Client Integration
One or more Protege GX controllers act as Modbus **servers**.	One Protege GX controller acts as a Modbus **client**.
The Modbus client reads and writes Protege GX inputs and outputs.	The controller reads and writes digital inputs (inputs), coils (outputs) and registers (data values) on connected Modbus server devices.
Communication over ethernet (TCP/IP).	Communication over RS-485.
Documented in Application Note 023: Protege GX Modbus Server Integration.	Documented in Application Note 353: Protege GX Modbus Client Integration.

This application note describes how to interface the Protege GX system controller to a Modbus network TCP/IP connection as a server device, to allow inputs and outputs connected to the Protege system to be read and controlled by the Modbus client. The integration is based on the protocol document V1.1b as described on the modbus.org website. For information on the Modbus protocol please consult the documentation.

Previously the Modbus client was referred to as the "master" and server devices were referred to as "slave" devices. This terminology has been deprecated by the Modbus Organization (see this press release).

How It Works

One or more Protege GX controllers can be connected to the Modbus network over ethernet. Each controller is identified on the Modbus network by a unique hexadecimal client address. This is configured in the customized Modbus service which facilitates communication between the controller and the Modbus network (see page 23).



Modbus TCP/IP

This implementation is Modbus TCP/IP. It is **not** Modbus over TCP/IP.

- Modbus TCP/IP means a Modbus TCP packet wrapped in a TCP packet.
- Modbus over TCP/IP means a Modbus RTU packet wrapped in a TCP packet.

The two are not compatible. They transfer like data but have different structures. Please ensure you are familiar with the protocol differences, and that any device that is communicating with the Protege GX system controller is using the same method of communication.

Prerequisites

Connecting a Protege GX system controller to a Modbus network requires the following firmware prerequisite.

Component	Version	Notes
Protege GX Controller	2.08.1342 or higher	

Supported Functionality

In simple terms, the Modbus network operates by reading from and writing to assigned objects. In this integration that equates to reading the status of inputs and outputs, and activating or deactivating outputs.

Function	Type	Value	Description
Read	Status	0	Input/Output is Off
	Status	1	Input/Output is On
Write	Control	0	Deactivate Output
	Control	1	Activate Output

Notes and Limitations

- You can set the number of items to read or write to. For example, you can specify a start and the count of coils.
- Since the controller has a limited buffer and each request consumes a certain amount of processing time, the maximum number of items is limited to an operationally sustainable size.
- Exceeding the total number of requested objects will result in an exception being returned to your calling application or device.
- Using an unsupported function code will return the appropriate exception.
- The python scripts provided in the appendix of this document detail the exceptions and provide examples where an exception is generated due to a malformed call.
- Writing past the limit of the registers or outside the bounds of the designated address space for multiple read or write will result in an exception being returned. The maximum number of objects is provided in the table below. For example, the maximum number of coils is 512, which equates to 64 bytes.
- Writing to multiple coils is supported. However, given the potential loading that this can cause it is limited to 16 coils in each call.
- Activating outputs will result in the output generating an event, if programmed to do so. Pay attention to the number of events being generated to ensure you maintain performance of your controller. To improve the user experience, disable events for outputs that you are activating frequently.

ICT provides a number of third-party connection options including the automation and control protocol, direct DLL calling using the REST API, and server-based SOAP calls through our open API. If the Modbus protocol does not provide the additional functionality required please investigate these options or get in touch with us so we can assist you with your integration needs.

Supported Modbus Function Codes

The following table identifies which Modbus function codes are supported by the integration, and the maximum number of items that can be requested by each supported function.

Function Type			Function	Function Code	Supported	Maximum
Data Access	Bit Access	Physical Discrete Inputs	Read Discrete Inputs	2	✓	512
		Physical and Virtual Coils/Outputs	Read Coils	1	✓	512
			Write Single Coil	5	✓	1
			Write Multiple Coils	15	✓	16
	16-Bit Access	Physical Input Registers	Read Input Registers	4	✗	N/A
		Internal Registers or Physical Output Registers	Read Multiple Holding Registers	3	✗	N/A
			Write Single Holding Register	6	✗	N/A
			Write Multiple Holding Registers	16	✗	N/A
			Read/Write Multiple Registers	23	✗	N/A
			Mask Write Register	22	✗	N/A
			Read FIFO Queue	24	✗	N/A
	File Record Access		Read File Record	20	✗	N/A
			Write File Record	21	✗	N/A
Diagnostics			Read Exception Status	7	✗	N/A
			Diagnostic	8	✗	N/A
			Get Com Event Counter	11	✗	N/A
			Get Com Event Log	12	✗	N/A
			Report Server ID	17	✗	N/A
			Read Device Identification	43	✗	N/A
Other			Encapsulated Interface Transport	43	✗	N/A

About Modbus

Modbus devices communicate using a client-server architecture in which only one device (the client) can initiate transactions (called queries). The other devices (servers) respond by supplying the requested data to the client, or by taking the action requested in the query.

A server is any peripheral device which processes information and sends it to the client using the Modbus protocol. The Protege GX controller operates as a server device on the Modbus network, while a typical client device is a host computer running appropriate application software.

Clients can address individual servers or initiate a broadcast message to all servers. Servers return a response to all queries addressed to them individually, but do not respond to broadcast queries. A client's query consists of a server address (or broadcast address), a function code defining the requested action, any required data, and an optional error checking field, depending on the physical connection being used. A server's response consists of fields confirming the action taken, any data to be returned, and optionally an error checking field.

Note that the query and response both include a device address, plus a function code, plus applicable data, and optional error checking field. If no error occurs the server's response contains the data requested. If an error occurs in the query received, or if the server is unable to perform the action requested, the server will return an exception message as its response (see Modbus Exceptions on the modbus.org website).

For more information, go to www.modbus.org.

Modbus Register Maps

Modbus functions operate on register map registers to monitor, configure, and control module I/O. A register map is provided in this document for the Protege GX controller. You will also find it helpful to refer to the register map as you review the functionality that is required to be integrated with the Protege GX controller.

Modbus registers are organized into reference types identified by the leading number of the reference address.

Register Reference Description

Address	Reference	Description
0xxxxx	Read/Write Discrete Outputs or Coils	A 0x reference address is used to drive output data to a digital output channel.
1xxxxx	Read Discrete Inputs	The ON/OFF status of a 1x reference address is controlled by the corresponding digital input channel.

Register Configuration

The following register map defines each coil and input that is available in the Protege GX controller. Any registers that are not defined should be treated as reserved locations and not used for any other purpose.

Map Structure

Output Coils Address Map

The following table summarizes the output coil addressing register (see page 12).

Protege Module Output Coils	Modbus Coil Address Range
Controller Output Coils	00001-00004
LCD Keypad Output Coils	00005-00516
Input Expander Output Coils	00517-01028
Reader Expander Output Coils	01029-02052
Output Expander Output Coils	02053-04100
Analog Input / Output Expander Output Coils	04101-04612

Inputs Address Map

The following table summarizes the input addressing register (see page 15).

Protege Module Inputs	Modbus Input Address Range
Controller Inputs	00001-00016
LCD Keypad Inputs	00017-00528
Input Expander Inputs	00529-02576
Reader Expander Inputs	02577-03600

Trouble Inputs Address Map

The following table summarizes the trouble input addressing register (see page 18).

Protege Module Trouble Inputs	Modbus Input Address Range
Controller Trouble Inputs	10001-10064
LCD Keypad Trouble Inputs	10065-11088
Input Expander Trouble Inputs	11089-13136
Reader Expander Trouble Inputs	13137-15184
Output Expander Trouble Inputs	15185-16208
Analog Input / Output Expander Trouble Inputs	16209-17232

Protege Module Models and Mapping

The register mapping is fixed and applies to all Protege modules. This includes current DIN rail modules and legacy PCB devices. Due to changes in hardware over time devices may have a different number of inputs or outputs than defined by the map structure, but the structure still applies and the sequence must be maintained for each device.

For example, for 8 output expanders only the first 8 Modbus coil numbers are used (2053 to 2060) but the second 8 output expander must still start at coil number 2069, and so on. The number of supported modules remains, whether or not the available addresses are used. If an I/O is no longer used, such as controller CP001:02, the mapping remains as though the output was still present.

Output Coils Address Map

Modbus coils are boolean values (1 or 0) that are typically used to represent an output device. Coils can be controlled (activate/deactivate) and can return the status of the output (on/off).

The following section outlines the Modbus coil address mapping for the outputs on supported Protege modules. A number of model configurations are supported for each device type, and the number of outputs available and their function depends on the model. For specific output assignments, see the appropriate installation manual.

Controller Output Coils

This mapping will allow the control and status checking of 4 onboard outputs on the controller.

Multiple controllers may be connected on the Modbus network. The same mapping applies to each controller.

Output Number	Output	Read/Write	Modbus Coil Number
CP001:01	Output 1	RD+WR	00001
CP001:02	Output 2	RD+WR	00002
CP001:03	Output 3	RD+WR	00003
CP001:04	Output 4	RD+WR	00004

LCD Keypad Output Coils

This mapping allows the control and status checking of outputs 1 to 4 on the first 128 registered keypads.

Output Number	Output	Read/Write	Modbus Coil Number
KP001:01	Output 1	RD+WR	00005
KP001:02	Output 2	RD+WR	00006
KP001:03	Output 3	RD+WR	00007
KP001:04	Output 4	RD+WR	00008
KP002:01	Output 1	RD+WR	00009
|||	|||	||	||
KP128:04	Output 4	RD+WR	00516

Input Expander Output Coils

This mapping allows the control and status checking of outputs 1 to 4 on the first 128 registered input expanders.

Output Number	Output	Read/Write	Modbus Coil Number
ZX001:01	Output 1	RD+WR	00517
ZX001:02	Output 2	RD+WR	00518
ZX001:03	Output 3	RD+WR	00519
ZX001:04	Output 4	RD+WR	00520
ZX002:01	Output 1	RD+WR	00521
| | |	| | |	| |	| |
ZX128:04	Output 4	RD+WR	01028

Reader Expander Output Coils

This mapping allows the control and status checking of outputs 1 to 8 on the first 128 registered reader expanders.

Output Number	Output	Read/Write	Modbus Coil Number
RD001:01	Output 1	RD+WR	01029
RD001:02	Output 2	RD+WR	01030
RD001:03	Output 3	RD+WR	01031
RD001:04	Output 4	RD+WR	01032
RD001:05	Output 5	RD+WR	01033
RD001:06	Output 6	RD+WR	01034
RD001:07	Output 7	RD+WR	01035
RD001:08	Output 8	RD+WR	01036
RD002:01	Output 1	RD+WR	01037
| | |	| | |	| |	| |
RD128:08	Output 8	RD+WR	02052

Output Expander Output Coils

This mapping allows the control and status checking of outputs 1 to 16 on the first 128 registered output expanders.

Output Number	Output	Read/Write	Modbus Coil Number
PX001:01	Output 1	RD+WR	02053
PX001:02	Output 2	RD+WR	02054
PX001:03	Output 3	RD+WR	02055
PX001:04	Output 4	RD+WR	02056
PX001:05	Output 5	RD+WR	02057
PX001:06	Output 6	RD+WR	02058
PX001:07	Output 7	RD+WR	02059
PX001:08	Output 8	RD+WR	02060
PX001:09	Output 9	RD+WR	02061
PX001:10	Output 10	RD+WR	02062
PX001:11	Output 11	RD+WR	02063
PX001:12	Output 12	RD+WR	02064
PX001:13	Output 13	RD+WR	02065
PX001:14	Output 14	RD+WR	02066
PX001:15	Output 15	RD+WR	02067
PX001:16	Output 16	RD+WR	02068
PX002:01	Output 1	RD+WR	02069
¦¦¦	¦¦¦	¦¦	¦¦
PX128:16	Output 16	RD+WR	04100

Analog Input / Output Expander Output Coils

This mapping allows the control and status checking of outputs 1 to 4 on the first 128 registered analog expanders.

Output Number	Output	Read/Write	Modbus Coil Number
AE001:01	Output 1	RD+WR	04101
AE001:02	Output 2	RD+WR	04102
AE001:03	Output 3	RD+WR	04103
AE001:04	Output 4	RD+WR	04104
AE002:01	Output 1	RD+WR	04105
¦¦¦	¦¦¦	¦¦	¦¦
AE128:04	Output 4	RD+WR	04612

Inputs Address Map

Inputs are boolean values (1 or 0) that are read-only and will return the status of an input (open/closed). Where an input is in any state other than open or closed it will be represented as open.

If an input address is requested that is not registered on the system or is outside the bounds of the input range an exception will be generated.

The following section outlines the Modbus input address mapping for the inputs on supported Protege modules. A number of model configurations are supported for each device type, and the number of inputs available depends on the model. For specific input assignments, see the appropriate installation manual.

Controller Inputs

This mapping will allow the status of 16 onboard inputs on the controller to be read.

Multiple controllers may be connected on the Modbus network. The same mapping applies to each controller.

Input Number	Input	Read/Write	Modbus Input Number
CP001:01	Input 1	RD	00001
CP001:02	Input 2	RD	00002
CP001:03	Input 3	RD	00003
CP001:04	Input 4	RD	00004
CP001:05	Input 5	RD	00005
CP001:06	Input 6	RD	00006
CP001:07	Input 7	RD	00007
CP001:08	Input 8	RD	00008
CP001:09	Input 9	RD	00009
CP001:10	Input 10	RD	00010
CP001:11	Input 11	RD	00011
CP001:12	Input 12	RD	00012
CP001:13	Input 13	RD	00013
CP001:14	Input 14	RD	00014
CP001:15	Input 15	RD	00015
CP001:16	Input 16	RD	00016

LCD Keypad Inputs

This mapping allows the status of inputs 1 to 4 on the first 128 registered keypads to be read.

Input Number	Input	Read/Write	Modbus Input Number
KP001:01	Input 1	RD	00017
KP001:02	Input 2	RD	00018
KP001:03	Input 3	RD	00019
KP001:04	Input 4	RD	00020
KP002:01	Input 1	RD	00021
| | |	| | |	|	| |
KP128:04	Input 4	RD	00528

Input Expander Inputs

This mapping allows the status of inputs 1 to 16 on the first 128 registered input expanders to be read.

Input Number	Input	Read/Write	Modbus Input Number
ZX001:01	Input 1	RD	00529
ZX001:02	Input 2	RD	00530
ZX001:03	Input 3	RD	00531
ZX001:04	Input 4	RD	00532
ZX001:05	Input 5	RD	00533
ZX001:06	Input 6	RD	00534
ZX001:07	Input 7	RD	00535
ZX001:08	Input 8	RD	00536
ZX001:09	Input 9	RD	00537
ZX001:10	Input 10	RD	00538
ZX001:11	Input 11	RD	00539
ZX001:12	Input 12	RD	00540
ZX001:13	Input 13	RD	00541
ZX001:14	Input 14	RD	00542
ZX001:15	Input 15	RD	00543
ZX001:16	Input 16	RD	00544
ZX002:01	Input 1	RD	00545
| | |	| | |	|	| |
ZX128:16	Input 16	RD	02576

Reader Expander Inputs

This mapping allows the status of inputs 1 to 8 on the first 128 registered reader expanders to be read.

Input Number	Input	Read/Write	Modbus Input Number
RD001:01	Input 1	RD	02577
RD001:02	Input 2	RD	02578
RD001:03	Input 3	RD	02579
RD001:04	Input 4	RD	02580
RD001:05	Input 5	RD	02581
RD001:06	Input 6	RD	02582
RD001:07	Input 7	RD	02583
RD001:08	Input 8	RD	02584
RD002:01	Input 1	RD	02585
RD128:08	Input 8	RD	03600

Trouble Inputs Address Map

Trouble inputs are boolean values (1 or 0) that are read-only and will return the status of a trouble input (open/closed).

If an input address is requested that is not registered on the system or outside the bounds of the input range an exception will be generated.

The following section outlines the Modbus input address mapping for the trouble inputs on supported Protege modules. A number of model configurations are supported for each device type, and the number of trouble inputs available and their function depends on the model. For specific trouble input assignments, see the appropriate installation manual.

Controller Trouble Inputs

This mapping will allow the status of the controller's trouble inputs to be read.

Multiple controllers may be connected on the Modbus network. The same mapping applies to each controller.

Trouble Input Number	Trouble Input	Read / Write	Input Number
CP001:01	Trouble Input 1	RD	10001
CP001:02	Trouble Input 2	RD	10002
CP001:03	Trouble Input 3	RD	10003
CP001:04	Trouble Input 4	RD	10004
CP001:05	Trouble Input 5	RD	10005
CP001:06	Trouble Input 6	RD	10006
CP001:07	Trouble Input 7	RD	10007
CP001:08	Trouble Input 8	RD	10008
CP001:09	Trouble Input 9	RD	10009
CP001:10	Trouble Input 10	RD	10010
CP001:11	Trouble Input 11	RD	10011
CP001:12	Trouble Input 12	RD	10012
CP001:13	Trouble Input 13	RD	10013
CP001:14	Trouble Input 14	RD	10014
CP001:15	Trouble Input 15	RD	10015
CP001:16	Trouble Input 16	RD	10016
CP001:17	Trouble Input 17	RD	1007
CP001:18	Trouble Input 18	RD	10018
CP001:19	Trouble Input 19	RD	10019
CP001:20	Trouble Input 20	RD	10020
| | |	| | | |	|	| |
CP001:64	Trouble Input 64	RD	10064

LCD Keypad Trouble Inputs

This mapping allows the status of the trouble inputs to be read for the first 128 registered keypads.

Trouble Input Number	Trouble Input	Read / Write	Input Number
KP001:01	Trouble Input 1	RD	10065
KP001:02	Trouble Input 2	RD	10066
KP001:03	Trouble Input 3	RD	10067
KP001:04	Trouble Input 4	RD	10068
KP001:05	Trouble Input 5	RD	10069
KP001:06	Trouble Input 6	RD	10070
KP001:07	Trouble Input 7	RD	10071
KP001:08	Trouble Input 8	RD	10072
KP002:01	Trouble Input 1	RD	10073
⁞	⁞	⁞	⁞
KP128:08	Trouble Input 8	RD	11088

Input Expander Trouble Inputs

This mapping allows the status of the trouble inputs to be read for the first 128 registered input expanders.

Trouble Input Number	Trouble Input	Read / Write	Input Number
ZX001:01	Trouble Input 1	RD	11089
ZX001:02	Trouble Input 2	RD	11090
ZX001:03	Trouble Input 3	RD	11091
ZX001:04	Trouble Input 4	RD	11092
ZX001:05	Trouble Input 5	RD	11093
ZX001:06	Trouble Input 6	RD	11094
ZX001:07	Trouble Input 7	RD	11095
ZX001:08	Trouble Input 8	RD	11096
ZX001:09	Trouble Input 9	RD	11097
ZX001:10	Trouble Input 10	RD	11098
ZX001:11	Trouble Input 11	RD	11099
ZX001:12	Trouble Input 12	RD	11100
ZX001:13	Trouble Input 13	RD	11101
ZX001:14	Trouble Input 14	RD	11102
ZX001:15	Trouble Input 15	RD	11103
ZX001:16	Trouble Input 16	RD	11104
ZX002:01	Trouble Input 1	RD	11105
| | |	| | | |	|	| |
ZX128:16	Trouble Input 16	RD	13136

Reader Expander Trouble Inputs

This mapping allows the status of the trouble inputs to be read for the first 128 registered reader expanders.

Trouble Input Number	Trouble Input	Read / Write	Input Number
RD001:01	Trouble Input 1	RD	13137
RD001:02	Trouble Input 2	RD	13138
RD001:03	Trouble Input 3	RD	13139
RD001:04	Trouble Input 4	RD	13140
RD001:05	Trouble Input 5	RD	13141
RD001:06	Trouble Input 6	RD	13142
RD001:07	Trouble Input 7	RD	13143
RD001:08	Trouble Input 8	RD	13144
RD001:09	Trouble Input 9	RD	13145
RD001:10	Trouble Input 10	RD	13146
RD001:11	Trouble Input 11	RD	13147
RD001:12	Trouble Input 12	RD	13148
RD001:13	Trouble Input 13	RD	13149
RD001:14	Trouble Input 14	RD	13150
RD001:15	Trouble Input 15	RD	13151
RD001:16	Trouble Input 16	RD	13152
RD002:01	Trouble Input 1	RD	13153
RD128:16	Trouble Input 16	RD	15184

Output Expander Trouble Inputs

This mapping allows the status of the trouble inputs to be read for the first 128 registered output expanders.

Trouble Input Number	Trouble Input	Read / Write	Input Number
PX001:01	Trouble Input 1	RD	15185
PX001:02	Trouble Input 2	RD	15186
PX001:03	Trouble Input 3	RD	15187
PX001:04	Trouble Input 4	RD	15188
PX001:05	Trouble Input 5	RD	15189
PX001:06	Trouble Input 6	RD	15190
PX001:07	Trouble Input 7	RD	15191
PX001:08	Trouble Input 8	RD	15192
PX002:01	Trouble Input 1	RD	15193
PX128:08	Trouble Input 8	RD	16208

Analog Input / Output Expander Trouble Inputs

This mapping allows the status of the trouble inputs to be read for the first 128 registered analog expanders.

Trouble Input Number	Trouble Input	Read / Write	Input Number
AE001:01	Trouble Input 1	RD	16209
AE001:02	Trouble Input 2	RD	16210
AE001:03	Trouble Input 3	RD	16211
AE001:04	Trouble Input 4	RD	16212
AE001:05	Trouble Input 5	RD	16213
AE001:06	Trouble Input 6	RD	16214
AE001:07	Trouble Input 7	RD	16215
AE001:08	Trouble Input 8	RD	16216
AE002:01	Trouble Input 1	RD	16217
AE128:08	Trouble Input 8	RD	17232

Programming the Service

This integration uses a customized service which facilitates communication between a Protege GX controller and the Modbus network.

A separate service is required for each Protege GX controller connected to the Modbus network, each with a unique client address configuration to identify the specific controller on the Modbus network.

Before connecting to a live Modbus network ensure the server ID and related information is accurate and that you have tested the connection. The Python validation scripts provided in the appendix (see next page) will allow you to communicate with the Modbus service and verify the points and registers prior to deployment to a live network.

1. To configure the Protege GX Modbus service, navigate to **Programming | Services**.
2. In the toolbar, select the controller that will be connected to the Modbus network.

   If multiple controllers are to be connected they will each require their own Modbus service configured.

3. Click **Add** to create the new service, and give the service an appropriate **Name**, such as Modbus Service.
4. Set the **Service type** to Modbus.
5. Set the **Service mode** to 1 - Start with controller OS.
6. In the **General** tab, set the Modbus service Configuration properties.
   - **Port number**: Must be set to TCP/IP.

     Once the service is started port 502 will be opened on the controller. This is the default port Modbus servers will communicate on.

   - **Client address**: The device address for the controller in the Modbus communication network. This should be a unique hexadecimal number which is not 0x00 or 0xFF. The client address is typically provided by the automation company or SCADA system that the controller will be connected to.

     If multiple Protege GX controllers are connected they each require their own unique client address.

   - **Poll time**: This field defines the maximum length of time (in seconds) expected between polls from the Modbus client. For example, if the poll time is set to 60 the controller will expect a poll every 60 seconds. If there is no poll an error will be logged in the event log and the **Output / Output group turns on when polling fails** will be activated.
   - **Output / Output group turns on when polling fails**: This output or output group is activated when the **Poll time** set above expires with no polling messages received. It is deactivated when the Modbus service completes a valid communication. Use this option to notify users that there is an issue in the Modbus system.
7. Select the required **Options** for the Modbus service.
   - **Log communication events**: When this option is enabled, events will be logged for all Modbus communications. This option may be useful for initial configuration and troubleshooting but should be disabled during normal operation to save event storage.
   - **Log communication errors**: When this option is enabled, events will be logged for all Modbus communication errors.
   - **Integers as big endian**: The default method of sending multi byte numbers is Little Endian (least significant byte first). With this option selected, multi byte numbers will be sent as Big Endian (most significant byte first).
   - **Use remote register variables**: This is a legacy option that has no effect.
   - **Enable coil input translation**: This is a legacy option that has no effect.
8. Click **Save**.
9. Download the changes to the controller and start the service

Appendix: Python Validation Scripts

To assist with connectivity and validation a series of Python scripts are provided, which allow you to exercise the ModBUS TCP/IP connection.

ICT does not directly support the Python libraries, however there is a wealth of information available by searching for Python examples.

When contacting ICT Technical Support for assistance you will be asked to execute the scripts provided. Because the connectivity options are limitless, using the scripts ensures that we are working to a known set of parameters.

The Python script examples use the following library: https://pymodbustcp.readthedocs.io/

This code is provided by Integrated Control Technology as is with no warranty or liability. The code is for the sole use and validation of the Modbus services available on ICT products and controllers.

Controller Input

```
# This code is provided by Integrated Control Technology Ltd as is with no
warranty or liability.
# The code is for the sole use and validation of the Modbus services available
on ICT products and controllers.

# Below will get the first 16 coils from the controller. It will then activate
and deactivate the third coil from address 0, this is relay 1 on the
controller, every two seconds.
#
# You can control relay 2 from another application and see the status change
while relay 1 is being controlled through Modbus.

from pyModbusTCP.client import ModbusClient
import time
SERVER_HOST = "192.168.1.2"
SERVER_PORT = 502
SERVER_U_ID = 1

try:
    c = ModbusClient(host=SERVER_HOST, port=SERVER_PORT, auto_open=True,
debug=False)
except ValueError:
    print("Error with host or port params")

# Uncomment this line to see debug messages being sent over the Modbus link
# c.debug(True)

try:
    while True:
        # read 16 bits at address 0, store result in bits list
        # first 4 bits are the controller outputs and then keypads 1-3.
        bits = c.read_discrete_inputs(0, 16)
        # if success display data
        if bits:
```

```
            print("Data ", end = '')
            for x in bits:
                if x == True:
                    print(1, end = '')
                else:
                    print(0, end = '')
            print("");
        else:
            print('unable to read inputs')
            print(c.last_except_as_txt)

    # sleep 2s before next polling of the relays
        time.sleep(2)

except KeyboardInterrupt:
    pass

c.close();
print("closing port")

print("done")
```

Controller Output

```
# This code is provided by Integrated Control Technology Ltd as is with no
warranty or liability.
# The code is for the sole use and validation of the Modbus services available
on ICT products and controllers.

# Below will get the first 16 coils from the controller. It will then activate
and deactivate the third coil from address 0, this is relay 1 on the
controller, every two seconds.
#
# You can control relay 2 from another application and see the status change
while relay 1 is being controlled through Modbus.

from pyModbusTCP.client import ModbusClient
import time
SERVER_HOST = "192.168.1.2"
SERVER_PORT = 502
SERVER_U_ID = 1

try:
    c = ModbusClient(host=SERVER_HOST, port=SERVER_PORT, auto_open=True,
debug=False)
except ValueError:
    print("Error with host or port params")

# Uncomment this line to see debug messages being sent over the Modbus link
# c.debug(True)

try:
```

```
    while True:
        # read 16 bits at address 0, store result in bits list
        # first 4 bits are the controller outputs and then keypads 1-3.
        bits = c.read_coils(0, 16)
        # if success display data
        if bits:
            print("Data ", end = '')
            for x in bits:
                if x == True:
                    print(1, end = '')
                else:
                    print(0, end = '')
            print("");
        else:
            print('unable to read coils')
            print(c.last_except_as_txt)

        if bits[2]:
            c.write_single_coil(2,0)
        else:
            c.write_single_coil(2,1)
    # sleep 2s before next polling of the relays
        time.sleep(2)

except KeyboardInterrupt:
    pass

c.close();
print("closing port")

print("done")
```

Controller Output - Multiple

```
# This code is provided by Integrated Control Technology Ltd as is with no
warranty or liability.
# The code is for the sole use and validation of the Modbus services available
on ICT products and controllers.

# *** read multiple coils ***
# *** write multiple coils ***
#
# Below will get the first 16 coils from the controller. It will then activate
and deactivate the third coil from address 0, this is relay 1 on the
controller (bit[2] in the response), doing this every two seconds.
#
# You can control relay 2 from another application and see the status change
while relay 1 is being controlled through Modbus.
#
# The functionality below is identical to that of the single coil write and
multi coil read however this uses the multiple coil write method.
```

```python
from pyModbusTCP.client import ModbusClient
import time
SERVER_HOST = "192.168.1.2"
SERVER_PORT = 502
SERVER_U_ID = 1

try:
    c = ModbusClient(host=SERVER_HOST, port=SERVER_PORT, auto_open=True,
debug=False)
except ValueError:
    print("Error with host or port params")

# Uncomment this line to see debug messages being sent over the Modbus link
# c.debug(True)

try:
    while True:
        # read 16 bits at address 0, store result in bits list
        # first 4 bits are the controller outputs and then keypads 1-3.
        bits = c.read_coils(0, 16)
        # if success display data
        if bits:
            print("Data ", end = '')
            for x in bits:
                if x == True:
                    print(1, end = '')
                else:
                    print(0, end = '')
            print("");
        else:
            print('unable to read coils')
            print(c.last_except_as_txt)
        #use bit 2 but could be any bit
        if bits[2]:
            bits[2] = False
        else:
            bits[2] = True

        res = c.write_multiple_coils(0,bits)
        if not res:
            print('unable to write coils')
            print(c.last_except_as_txt)
    # sleep 2s before next polling of the relays
        time.sleep(2)

except KeyboardInterrupt:
    pass

c.close();
print("closing port")

print("done")
```

Controller Test Limits

```
# This code is provided by Integrated Control Technology Ltd as is with no
warranty or liability.
# The code is for the sole use and validation of the Modbus services available
on ICT products and controllers.

# This file is designed to validate the exceptions in various calls and ensure
that the return codes are correct. This will not actually perform any action
as ALL of the calls here are illegal and create an exception of some kind.

from pyModbusTCP.client import ModbusClient
import time
SERVER_HOST = "192.168.1.2"
SERVER_PORT = 502
SERVER_U_ID = 1

try:
    c = ModbusClient(host=SERVER_HOST, port=SERVER_PORT, auto_open=True,
debug=False)
except ValueError:
    print("Error with host or port params")

# Uncomment this line to see debug messages being sent over the Modbus link
# c.debug(True)

# Used to trigger a multi write failure
toolong = [False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False ]
somebits = [False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False ]

try:

# Multi coil read
    print('Request number of coils over buffer allowance')
    bits = c.read_coils(0, 513)
    # if success display data
    if bits:
        print("************** FAILED VALIDATION **************");
    else:
        print(c.last_except_as_txt)

    print('Request good number of coils however starting coil exceeds end of
allowable range')
    bits = c.read_coils(4600, 16)  # see application note for offsets
    # if success display data
    if bits:
        print("************** FAILED VALIDATION **************");
    else:
        print(c.last_except_as_txt)
```

```
# Multi input read
# These cover trouble inputs as well so we want to check in the gap below
address 10000
    print('Request number of inputs over buffer allowance')
    bits = c.read_discrete_inputs(0, 513)
    # if success display data
    if bits:
        print("************** FAILED VALIDATION **************");
    else:
        print(c.last_except_as_txt)

    print('Request good number of inputs however starting input exceeds end of
input zone range')
    bits = c.read_discrete_inputs(3585, 16)  # see application note for
offsets
    # if success display data
    if bits:
        print("************** FAILED VALIDATION **************");
    else:
        print(c.last_except_as_txt)

    print('Request good number of inputs however address is past end of
trouble inputs ')
    bits = c.read_discrete_inputs(17217, 16)  # see application note for
offsets
    # if success display data
    if bits:
        print("************** FAILED VALIDATION **************");
    else:
        print(c.last_except_as_txt)

# Coil write
    print('Write single coil outside bounds of output address')
    res = c.write_single_coil(4613,1)
    if not res:
        print(c.last_except_as_txt)
    else:
        print("************** FAILED VALIDATION **************");

# Multi coil write
    print('Write more coils in multiwrite than allowed')
    res = c.write_multiple_coils(0,toolong)
    if not res:
        print(c.last_except_as_txt)
    else:
        print("************** FAILED VALIDATION **************");

    print('Write the correct number of coils but outside the address range')
    res = c.write_multiple_coils(4600,somebits)
    if not res:
        print(c.last_except_as_txt)
    else:
```

```
        print("************** FAILED VALIDATION **************");

except KeyboardInterrupt:
    pass

c.close();
print("closing port")

print("done")
```