Honeywell

ControlEdge PLC
ControlEdge RTU
Release 174.1

ControlEdge Builder Protocol Configuration Reference Guide

RTDOC-X288-en-174A
December 2022

DISCLAIMER

This document contains Honeywell proprietary information. Information contained herein is to be used solely for the purpose submitted, and no part of this document or its contents shall be reproduced, published, or disclosed to a third party without the express permission of Honeywell International Sàrl.

While this information is presented in good faith and believed to be accurate, Honeywell disclaims the implied warranties of merchantability and fitness for a purpose and makes no express warranties except as may be stated in its written agreement with and for its customer.

In no event is Honeywell liable to anyone for any direct, special, or consequential damages. The information and specifications in this document are subject to change without notice.

Copyright 2022 – Honeywell International Sàrl

CONTENTS

CHAPTER

1 ABOUT THIS GUIDE

Revision history

Revision	Date	Description
A	December 2022	Initial release of this document

Intended audience

This documentation is intended for the following audience: users who plan, install, configure, operate, or maintain the ControlEdge™ 900 and 2020 controller and I/O modules running the eCLR (IEC 61131-3) execution environment.

Prerequisite skills

Knowledge of SCADA systems and experience of working in a Microsoft Windows environment are required.

Introduction to ControlEdge Technology

Item	Description
ControlEdge PLC	ControlEdge 900 controllers running the eCLR (IEC 61131-3) execution environment with PLC software options configured with ControlEdge Builder.
ControlEdge RTU	ControlEdge 2020 controllers running the eCLR (IEC 61131-3) execution environment with RTU software options configured with ControlEdge Builder.
ControlEdge UOC	ControlEdge 900 controllers running the Honeywell control execution environment (CEE) configured with Experion Control Builder.

Special terms

The following table describes some commonly used industry-wide and Honeywell-specific terminology:

Terminology	Description
CDA	Control Data Access
ControlEdge 900 controller OPC UA	OPC UA runs on ControlEdge 900 controller
DNP3	Distributed Network Protocol V3.0
EFM	Electronic Flow Measurement
Enron Modbus	An extension of standard Modbus supports for 32-bit Integer and Floating Point variables, and historical and flow data.
HART-IP	HART-IP extends the HART protocol to Ethernet connected nodes. This facilities host level systems and asset management applications to access and integrate measurement and device diagnostics information from HARTenabled field devices using the existing plant networking infrastructure.
Modbus	A communication protocol supports communication between Modbus slave devices and Modbus master devices via serial port or Ethernet port.
MQTT	Message Queuing Telemetry Transport, an open OASIS and ISO standard (ISO/IEC 20922) lightweight, publish-subscribe network protocol that transports messages between devices. The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections.
OPC	Open Platform Communications
OPC UA	OPC Unified Architecture
QoS	The Quality of Service (QoS) level is an agreement between the sender and the receiver of a message that defines the guarantee of delivery for a specific message. There are 3 QoS levels in MQTT: • At most once delivery (0); • At least once delivery (1); • Exactly once delivery (2).
SCADA	Supervisory Control and Data Acquisition
Sparkplug	Sparkplug provides an open and freely available specification for how Edge of Network (EoN) gateways or native MQTT enabled

Terminology	Description
	end devices and MQTT Applications communicate bi-directionally within an MQTT Infrastructure.
TLS	Transport Layer Security; TLS is a cryptographic protocol that provide communications security over a computer network.

Related documents

The following list identifies publications that may contain information relevant to the information in this document:

- Builder Software Installation User's Guide
- ControlEdge Builder Software Change Notice
- ControlEdge PLC and ControlEdge RTU Getting started
- ControlEdge Builder User's Guide
- ControlEdge 900 Platform Hardware Planning and Installation Guide
- ControlEdge 2020 Platform Hardware Planning and Installation Guide
- ControlEdge Builder Function and Function Block Configuration Reference Guide
- ControlEdge PLC and ControlEdge RTU Network and Security Planning Guide
- ControlEdge EtherNet/IP User's Guide
- ControlEdge RTU and PLC DNP3 Device Profile
- ControlEdge Bulk Configuration User's Guide
- Firmware Manager User Guide
- ControlEdge PLC PROFINET User's Guide
- ControlEdge RTU Electronic Flow Measurement User's Guide

CHAPTER

2 OVERVIEW

ControlEdge PLC and ControlEdge RTU supports various kinds of protocol configuration. See the following table for details:

Protocol	Description	Supported by
DNP3 Outstation	See DNP3 Outstation Configuration for more information.	ControlEdge PLC and ControlEdge RTU
DNP3 Master	See DNP3 Master Configuration for more information.	ControlEdge RTU
Modbus Slave	See Modbus Slave Configuration for more information.	ControlEdge PLC and ControlEdge RTU
HART-IP Sever	See HART Configuration for more information.	ControlEdge PLC and ControlEdge RTU
Enron Modbus Slave	See Enron Modbus Slave Configuration for more information.	ControlEdge RTU
Modbus Master	See Modbus Master Configuration for more information.	ControlEdge PLC and ControlEdge RTU
OPC UA Client	See OPC UA Client for more information.	ControlEdge PLC
OPC UA Server	See OPC UA Server for more information.	ControlEdge PLC
CDA Responder	See CDA Configuration for more information.	ControlEdge PLC
User Defined Protocol	See User Defined Protocol for more information.	ControlEdge PLC and ControlEdge

Protocol	Description	Supported by
		RTU
MQTT	See MQTT Configuration for more information.	ControlEdge RTU
Wireless I/O	See "Configuring Wireless I/O" in ControlEdge Builder User Guide for more information.	ControlEdge RTU
EtherNet/IP	See *EtherNet/IP User's Guide* for more information.	ControlEdge PLC
PROFINET	See *ControlEdge PLC PROFINET User Guide* for more information.	ControlEdge PLC

CHAPTER

3 HART CONFIGURATION

HART supports two functionalities.

- HART IP client (FDM) communication
- HART Function Block communication

The controller enables the HART IP client to exchange information with HART field devices connected to the AI/AO channels in the controller via a HART-IP Server. Multiple HART IP clients can be served by the controller at the same time. When the HART IP client builds a HART command request and sends it to the TCP/IP port of the HART-IP server, the HART-IP server responds to the HART IP client with information from the field device. Since it takes time for the controller to communicate with the field devices through onboard or remote I/O cards, a delayed response mechanism is implemented. The TCP /IP port of the HART-IP server is user-configurable and the default port number is 5094. The end user may change the port number if firewall configuration is required.

The controller enables HART function blocks to access to the HART field devices through HART-enabled AI/AO channels. Currently HART command 3, command 48 and command X are implemented.

Configuring a HART IP Server

A new project is created and a controller is added to the project in ControlEdge Builder. See "Creating a project" and "Connecting a controller" in *ControlEdge Builder User's Guide* for more details.

To set a controller as the HART IP Server:

1. From the Home Page, click **Configure Ethernet Ports** and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the details in the **IP Address**, **Subnet Mask** and **Gateway** fields.
3. Under **Protocol Binding**, select **HART IP** to bind HART IP to the Ethernet port.
4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.

5. Click **Configure Protocols** > **HART IP Server**, select the target Ethernet port and configure the port number in the **Port**. The default value is 5094.
6. From the Home Page, click **Configure I/O**, and configure the target AI or AO channel. For more information, see "Configuring I/O modules and channels" in the *ControlEdge Builder User's Guide*.
7. Select the **Enable** checkbox for HART, and click **Save**.
8. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
9. Click **Download** from the Home Page to load the configuration of HART IP to the controller.

Configuring a HART Function Block

From R151, two sets of HART function blocks are provided, HART and HART_V2. HART _V2 is recommended to be used.

Follow the instructions below to program the HART device for the project in **IEC Programming Workspace**.

To configure a HART function block:

1. From the **IEC Programming Workspace**, under the **Project Tree Window**, right-click **Logical POUs** and select **Insert** > **Program**.
2. Enter the **Name** for the new POU, and select the desired programming Language. For the following steps, FBD language is used as an example.
3. Click **OK** to insert the new POU in the project tree.
4. Add a **Task** as follows:
   a. Under **Physical Hardware**, right-click **Task** and select **Insert** > **Task**.
   b. Enter the **Name** and select the task type as **CYCLIC**, and click **OK**.
   c. In the **Task settings** dialog, configure the corresponding parameters.
   d. Click **OK**.
5. Right-click the task you have inserted, and select **Insert** > **Program instance**.
6. Enter a name in the **Program instance** field.

   The program instance must not be named “RTU” or “GlobalVariable”.

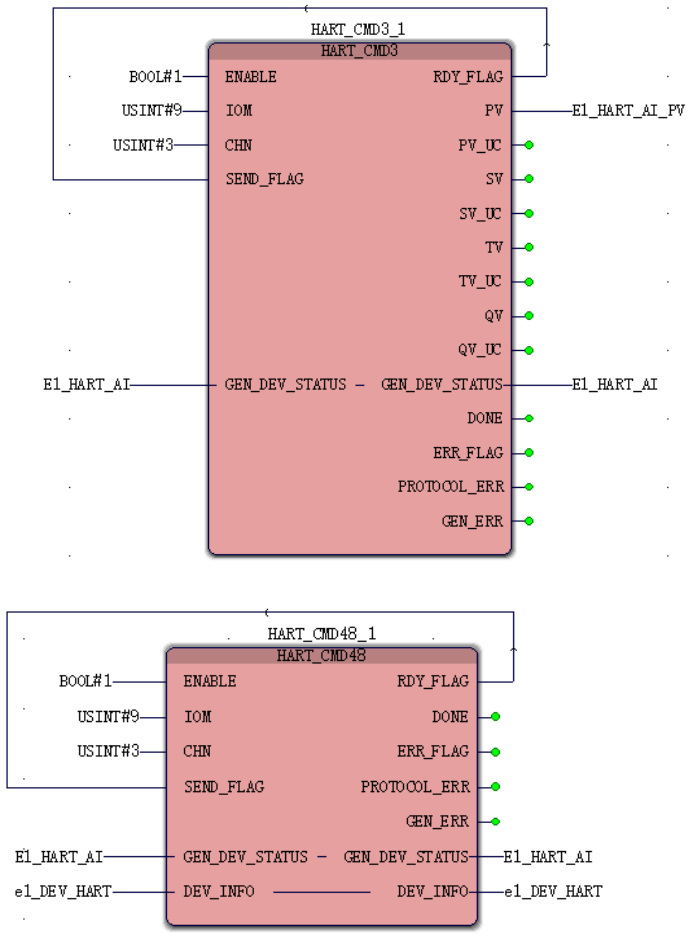
7. Select the program you want to associate from the **Program type** drop-down list. Click **OK**.
8. Right-click **Libraries** and select **Insert** > **Firmware Library**, select **hart.fwl** under the **HART** folder. Then click **Include**.
9. Under Logical POUs, double-click the code worksheet  of the program that you have inserted.
10. From the Edit Wizard, select **HART** from the **Group** list. There are three function blocks available for HART programming: HART_CMD3 and HART_CMD48 as well as HART_CMDx.
11. Drag the target function block into the workplace to display the function block.

    For more information about the function block, right-click it and select **Help on FB/FU** to display the embedded help.
12. Double-click the pin-outs of the function block to assign variables. The **Variable Properties** dialog appears.
13. Select the **Name**, **Data Type** and **Usage** from the list.

    - For the parameter **GEN_DEV_STATUS**, you should select **HAR_GEN_DEV_STATUS** from the **Data Type** list.
    - For the parameter **DEV_INFO**, you should select **HART_CMD48_DEV_INFO** from the **Data Type** list.
14. Assign Initial value and I/O address details.

    - For the parameter **IOM**, enter the target module number in the **Initial Value** field. For example, if the target module name is "Expansion I/O 01", enter "01".
    - For the parameter **CHN**, enter the target channel number in the **Initial Value** field.

15. Click **OK**. The workplace will appear as shown below.



16. Click **Make** from the toolbar to compile the programs.
17. Click **Download** from the toolbar to download the compiled programs of HART to the controller.

CHAPTER

4 DNP3 OUTSTATION CONFIGURATION

Configuring a DNP3 Outstation

> **ATTENTION:** DNP3 supports a maximum of 500 events per second.

1. From the Home Page, click **Configure Ethernet Ports** and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the details in the **IP Address**, **Subnet Mask** and **Gateway** fields.
3. Under **Protocol Binding**, select **DNP3 Outstation** to bind DNP3 Outstation to the Ethernet port.
4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.
5. Under **I/O and Communications** tab, click **Configure Protocols** > **DNP3 Outstation**.
6. Click **Add a Master**. The **Add DNP3 Master** dialog appears.
7. Select **Ethernet port** and **Master Index**.

   > **TIP:** Up to 5 DNP3 masters are supported for one Ethernet port.

8. Select **Enable Channel Redundancy** if required.

   > **NOTE:** This option is ONLY available for Ethernet port 1 ETH1.

9. Click **OK** to add a master.

   If you select **Enable Channel Redundancy**, both ports ETH1 and ETH2 appear. They share a single configuration form at ETH1.

10. In the **General** group, configure the following parameters.

Parameter	Description
Mapping	Select the required mapping table from the drop-down list. If the Mapping is empty, you must add a mapping table first. See "Adding a DNP3 Outstation Mapping table" in the *ControlEdge Builder User's Guide* for more information. **For redundant channel**, the same mapping table must be selected on multiple ports. For example, this could be used when a SCADA system communicates through 2 ports in a redundant arrangement. **For individual channel**: For R151 and before, one mapping table must be used for one port. Starting from R160, one mapping table can be used for multiple ports.
TCP Port	Configure TCP port number.
Master Address	Configure Master Address.
Controller Outstation Address	Configure Controller Outstation Address.
Enable Self Address	Select **Enable Self Address** for the controller to respond with its unique individual address, if a message is sent with the “Self Address”. If **Enable Self Address** is not selected, the controller will ignore the message sent to the “Self Address”.
Data Link Confirmation	**Never** is selected by default. It is not recommended to select **MultiFrag** or **Always** options. If you select **MultiFrag** or **Always**, ensure that the **Data Link Retries** and **Data Link Retry Timeout** are set.
Data Link Retries	It must be configured if **Data Link Confirmation** is selected as **MultiFrag** or **Always**. The maximum value is 255.

Parameter	Description
Data Link Retry Timout	It must be configured if **Data Link Confirmation** is selected as **MultiFrag** or **Always**. The maximum value is 3,600,000ms.

11. In the **Application Layer** group, if you select **Enable Unsolicited Responses**, the controller sends event data to SCADA without any request from SCADA. Unsolicited Response is an operation mode in which the outstation spontaneously transmits a response without a specific request for the data.

Parameter	Description
Send NULL Unsolicited Responses on Reconnect	The DNP3 driver sends a null unsolicited message upon reconnect once it is selected.
Maximum Hold Delay of Class1/Class2/Class3	The maximum hold delay is the maximum amount of time that the controller will wait after an event occurs before sending an unsolicited response. This setting allows the controller to queue several events before sending an unsolicited response, improving bandwidth usage at the expense of delayed communication. Minimum value: 0 ms and maximum value: 3,600,000ms.
Maximum Hold Count of Class1/Class2/Class3	The maximum hold count is the maximum number of events that may be queued before sending an unsolicited response. This setting allows the controller to send multiple events in a single message, improving bandwidth usage at the expense of delayed communication.
Unsolicited Response Retries	Enter the number of times the DNP3 driver attempts to send the unsolicited application fragment upon not receiving confirmation.

Parameter	Description
	The value can be 0 to 255.
Unsolicited Response Retry Delay	Enter the time intervals between retry to send the unsolicited response.
Delete Oldest Event on Event Overflow	According to requirement, select Delete Oldest Event when queue is full or not. If this option is checked, in case the DNP3 event buffer is full, then any new event overwrites the oldest event.
Validate Controller Outstation Address	If this option is checked, the controller only accepts data from some specific outstation addresses.
Keepalive Interval	Enter the interval that DNP3 outstation sends response to master station to make sure if the connection is normal.
Enable DNP3 Time Synchronization	Enable time synchronization from the DNP3 master. **NOTE:** Only one master can be enabled time synchronization.
DNP3 Time Synchronization Period	Select the time that the controller should indicate to SCADA that the time synchronization is required. **ATTENTION:** If you select DNP3 Time Sync here, you cannot enable Primary Server and Secondary Server under Miscellaneous > Configure Date/Time options at the same time, or else you cannot download your configuration.
Solicited Response Confirmation Timeout	Enter the time in milliseconds the DNP3 driver waits for confirmation for the sent solicited application fragment.
Unsolicited Response Confirmation Timeout	Enter the time in milliseconds the DNP3 driver waits for confirmation for the sent

Parameter	Description
	unsolicited application fragment. Maximum value for timeout is 3600000ms.
Select Before Operation (SBO) Timeout	Enter the time in milliseconds the DNP3 driver waits for SBO.
EFM Data Class	Select the corresponding class for EFM data to SCADA communication. Set the time interval for getting the EFM responses back to Experion from the controller through the DNP3 virtual terminal point. There are three options defined by the DNP3 master: • Class 1 • Class 2 • Class 3

12. In the **Default Variation** group, configure the default variation for each type of DNP3 point. Default variation defines the data format that is used by the controller to send data to the DNP3 Master, when the Master does not ask for a specific data variation.

Parameter	Description
Binary Input	Used to report the current value of a binary input point with three options: • Any variation • Packed format • Value with flags
Binary Input Event	Used to report events related to a binary input point with four options: • Any variation

Parameter	Description
	• Value without time • Value with absolute time • Value with relative time
Double-bit Binary Input	Used to report the current value of a double-bit binary input point with three options: • Any variation • Packed format • Value with flags
Double-bit Binary Input Event	Used to report events related to a double-bit binary input point with four options: • Any variation • Value without time • Value with absolute time • Value with relative time
Binary Output	Used to control or report the state of one or more binary output points with three options: • Any variation • Packed format • Status with flags
Binary Output Event	A Binary Output Event Object is an instance of a report for an outstation's corresponding Binary Output Static object. • Any variation • Status without time • Status with time
Binary Output Command Event	A Binary Output Command Event object reports that a command has been attempted on an outstation's corresponding binary output point. • Any variation

Parameter	Description
	• Status without time • Status with time
Counter	Used to report the current value of a counter point with five options: • Any variation • 32-bit integer with flag • 16-bit integer with flag • 32-bit integer without flag • 16-bit integer without flag
Frozen Counter	Used to report the value of a counter point captured at the instant when the count is frozen with seven options: • Any variation • 32-bit integer with flag • 16-bit integer with flag • 32-bit integer with flag. time • 16-bit integer with flag. time • 32-bit integer without flag • 16-bit integer without flag
Counter Event	Used to report the value of a counter point after the count has changed with five options: • Any variation • 32-bit integer with flag • 16-bit integer with flag • 32-bit integer with flag. time • 16-bit integer with flag. time
Frozen Counter Event	Used to report, as an event, the value of a counter point captured at the instant when the count is frozen.

Parameter	Description
	• Any variation • 32-bit integer with flag • 16-bit integer with flag • 32-bit integer with flag. time • 16-bit integer with flag. time
Analog Input	Used to report the current value of an analog input point with seven options: • Any variation • 32-bit integer with flag • 16-bit integer with flag • 32-bit integer without flag. time • 16-bit integer without flag. time • Single-precision float with flag • Double-precision float with flag
Analog Input Event	Used to report events related to an analog input point with nine options: • Any variation • 32-bit integer with time • 16-bit integer with time • 32-bit integer without time • 16-bit integer without time • Single-precision float with time • Double-precision float with time • Single-precision float without time • Double-precision float without time
Analog Input Deadband	Used to set and report the deadband value of an analog input point with four options: • Any variation

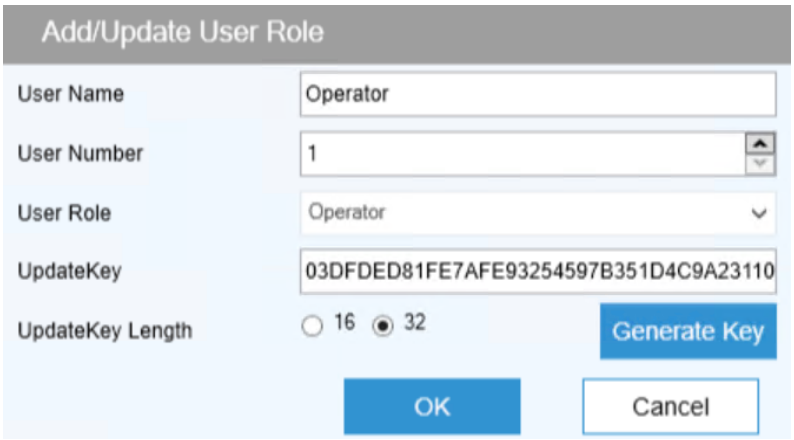
Parameter	Description
	• 16-bit integer • 32-bit integer • Single-precision float
Analog Output Status	Used to report the status of an analog output point with seven options: • Any variation • 32-bit integer with flag • 16-bit integer with flag • Single-precision float with flag • Double-precision float with flag
Analog Output Event	An Analog Output Event Object is an instance of a report for an outstation's corresponding Analog Output Status object. There are nine options: • Any variation • 32-bit integer with time • 16-bit integer with time • 32-bit integer without time • 16-bit integer without time • Single-precision float with time • Double-precision float with time • Single-precision float without time • Double-precision float without time
Analog Output Command Event	An Analog Output Command Event object reports that a command has been attempted on an outstation's corresponding Analog Output point. There are nine options: • Any variation • 32-bit integer with time • 16-bit integer with time

Parameter	Description
	• 32-bit integer without time • 16-bit integer without time • Single-precision float with time • Double-precision float with time • Single-precision float without time • Double-precision float without time

13. In the **Secure Authentication v5** tab, configure secure authentication, user role configure settings, critical function code, and MAC algorithm for DNP3 secure communication.

    i. Select **Enable Secure Authentication**.

    ii. In the **User Role Configure settings** tab, click **Add**. Add/Update User Role dialog box appears. See the following image.



See the following table for the parameter descriptions:

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>User Name</td><td>Enter a user name to quickly identify the user role.

NOTE: User Name must be unique.</td></tr>
<tr><td>User Number and User Role</td><td>Select the user number and user role from the drop-down list.

<table>
<tr><th>User Number</th><th>User Role</th></tr>
<tr><td>1</td><td>Operator</td></tr>
<tr><td>2</td><td>Engineer</td></tr>
<tr><td>3</td><td>Installer</td></tr>
<tr><td>4</td><td>Security Admin</td></tr>
<tr><td>5</td><td>Security Audit</td></tr>
<tr><td>6</td><td>RBACMNT</td></tr>
<tr><td>7</td><td>Single User</td></tr>
<tr><td>8</td><td>Viewer</td></tr>
</table>
</td></tr>
<tr><td>UpdateKey</td><td>Update key is a pre-shared key. Using the update key, the master can create a session and change the session key periodically..

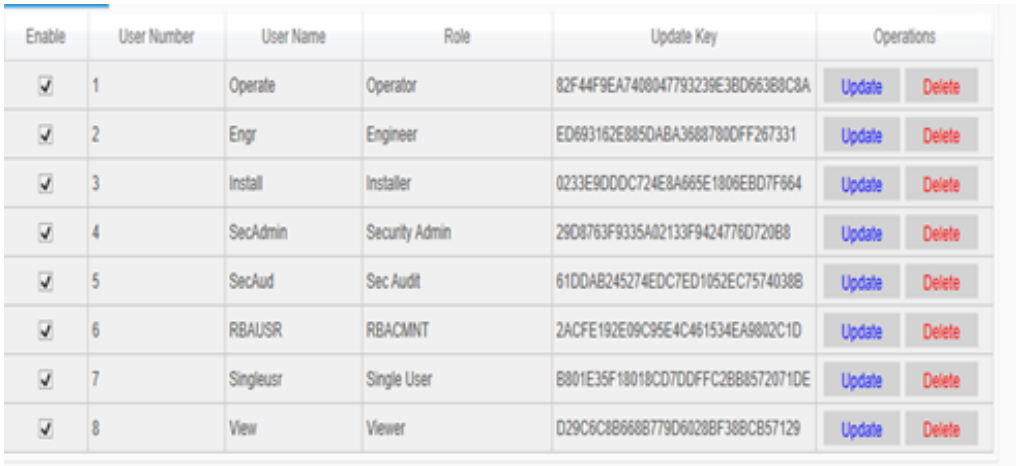
NOTE: Update key must match with master and RTU to enable the session and perform changes from the master.

If the key is not shared, click Generate Key to generate the new key and share the new key to the master.</td></tr>
<tr><td>UpdateKey Length</td><td>It is length of the update key. Select the Updatekey length as 16 or 32.

NOTE: At master and RTU, the Update Key</td></tr>
</table>

Parameter	Description
	must have same length.

a. Once all the parameters are configured, click **OK**. User roles are added. See the following image for reference.



Enable	User Number	User Name	Role	Update Key	Operations
☑	1	Operate	Operator	82F44F9EA7408047793239E3BD663B8C8A	Update Delete
☑	2	Engr	Engineer	ED693162E885DABA3688780DFF267331	Update Delete
☑	3	Install	Installer	0233E9DDDC724E8A665E1806EBD7F664	Update Delete
☑	4	SecAdmin	Security Admin	29D8763F9335A02133F9424776D720B8	Update Delete
☑	5	SecAud	Sec Audit	61DDAB245274EDC7ED1052EC7574038B	Update Delete
☑	6	RBAUSR	RBACMNT	2ACFE192E09C95E4C461534EA9802C1D	Update Delete
☑	7	Singleusr	Single User	B801E35F18018CD7DDFFC2BB8572071DE	Update Delete
☑	8	View	Viewer	D29C6C8B668B779D6028BF38BCB57129	Update Delete

b. To update the user role parameters, click **Update**. **Add/Update User Role** dialog box appears. Update the required user role parameters and click **OK**.

c. To delete the user role, click **Delete**. A confirmation dialog box appears and click **OK**.

iii. (Optional) Enable or Disable **Aggressive mode** as per the requirement.

> **NOTE:** By default, Aggressive mode is disabled.

See the following table for parameter description:

Parameter	Description
Aggressive mode	To reduce bandwidth usage, a responder attempting a critical operation may optionally “anticipate” the challenge and send the MAC Value in the same ASDU being protected. This practice is known as “aggressive mode”. It eliminates the challenge and reply messages. For this reason, aggressive mode is optional in

Parameter	Description
	IEC 62351-5.

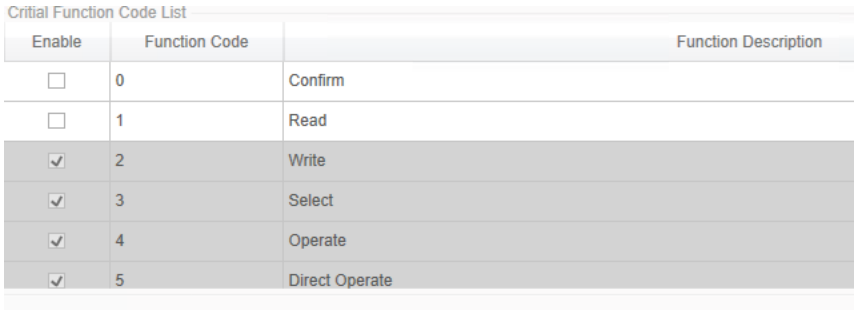
iv. Enter **Challenge Data Length**. Users can enter challenge data length ranging from 4 to 64.

v. In the **Critical Function Code List** tab, enable or disable the function code to define a function as critical or non critical.

- The greyed out and pre-selected functions are executed with a challenge response mechanism only. The remaining not selected functions are non-critical, and they can be enabled as critical functions if it is to be executed with a challenge response mechanism.

> **NOTE:** The greyed out and pre-selected functions can not be modified or defined as non-critical functions.

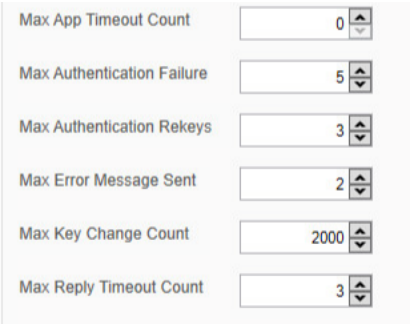
See the following image.



vi. Select the **MAC Algorithm** from drop down list.
Supported MAC algorithms:

- SHA1_4OCTET
- SHA1_8OCTET
- SHA1_10OCTET
- SHA256_8OCTET
- SHA256_16OCTET
- AESGMAC_12OCTET

vii. Enter **Max App Timeout Count**, **Max Authentication Failure**, **Max Authentication Rekeys**, **Max Error Message Sent**. **Max Key Change Count,** and **Max Reply Timeout Count**. See the following image.



See the following table for parameter description:

Parameter	Description
Max App Timeout Count	Number of app timeouts after which secure authentication failure event happens.
Max Authentication Failure	Number of authentication failures after which Rekey due to fail is incremented.
Max Authentication Rekeys	If exceeded, stop changing session keys due to authentication failure.
Max Error Message Sent	If exceeded, stop sending error message objects.
Max Key Change Count	Change session keys whenever a configured number of authentication ASDUs has been transmitted in either direction since last key change.
Max Reply Timeout Count	If exceeded, cancel the current transaction.

viii. In **Symmetric UpdateKey change method** tab, enable Symmetric UpdateKey change method.

a. Select **Authority Symmetric cert key** length as **16** or **32**.

b. Click **Generate Key**. Authority Symmetric Cert Key is generated.

See the following table for parameter description:

Parameter	Description
Session Key	Each user owns a set of session key which is used to authenticate data. Master generates it

Parameter	Description
	and periodically (Minutes to weeks) changes it on both sides.
Update Key	It is a pre-shared key. Using the update key, the master can create a session and change the session key periodically. **NOTE:** Update key must match with master and RTU to enable the session and perform changes from the master.
Authority Key	It is a pre-shared key, It must be matching to enable the master to change the update key at master side and replicate the same key at RTU side. CHANGED IN MONTHS or YEARS.

14. Select **Flash** or **SD card** from the drop-down list besides **Save DNP3 Events to:**.
    - If you want to save DNP3 events to an SD card, you must allocate the space for DNP3 events first. See "Preparing SD card" in *ControlEdge Builder User's Guide* for more information.
    - Up to 200,000 DNP3 events can be saved to Flash per ControlEdge 2020 controller.
    - Up to 100,000 DNP3 events can be saved to Flash per ControlEdge 900 controller.
    - Up to 500,000 DNP3 events can be saved to an SD card per controller.
15. Click **Save**.
16. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
17. Click **Download** from the Home Page to load the configuration of the DNP3 Outstation to the controller.

CHAPTER

5 DNP3 MASTER CONFIGURATION

DNP3 Master is used for communication between the controller and third-party DNP3 outstation devices over Ethernet. You need to bind the protocol to the Ethernet port of your controller and program the DNP3 Master for the project.

Configuring a DNP3 Master

A new project should be created and a controller should be added to the project opened in ControlEdge Builder. See "Creating a project" and "Connecting a controller" in *ControlEdge Builder User's Guide* for more details.

To set a controller as a DNP3 Master:

1. From the Home Page, click **Configure Ethernet Ports** and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the details in the **IP Address**, **Subnet Mask** and **Gateway** fields.
3. Under **Protocol Binding**, select **DNP3 Master** to bind DNP3 Master to the Ethernet port.
4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.
5. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
6. Click **Download** from the Home Page to load the configuration of the DNP3 Master to the controller.

Programming a DNP3 Master

Follow the instructions below to program DNP3 Master for the project in **IEC Programming Workspace**.

1. Right-click **Logical POUs** and select **Insert** > **Program**. Then enter the name, and click **OK**. For the following steps, FBD language is used as an example.
2. Add a **Task** as follows:

a. Under **Physical Hardware**, right-click **Task** and select **Insert** > **Task**.

b. In the pop-up window, enter the name. Select the task type as **CYCLIC**, and click **OK**.

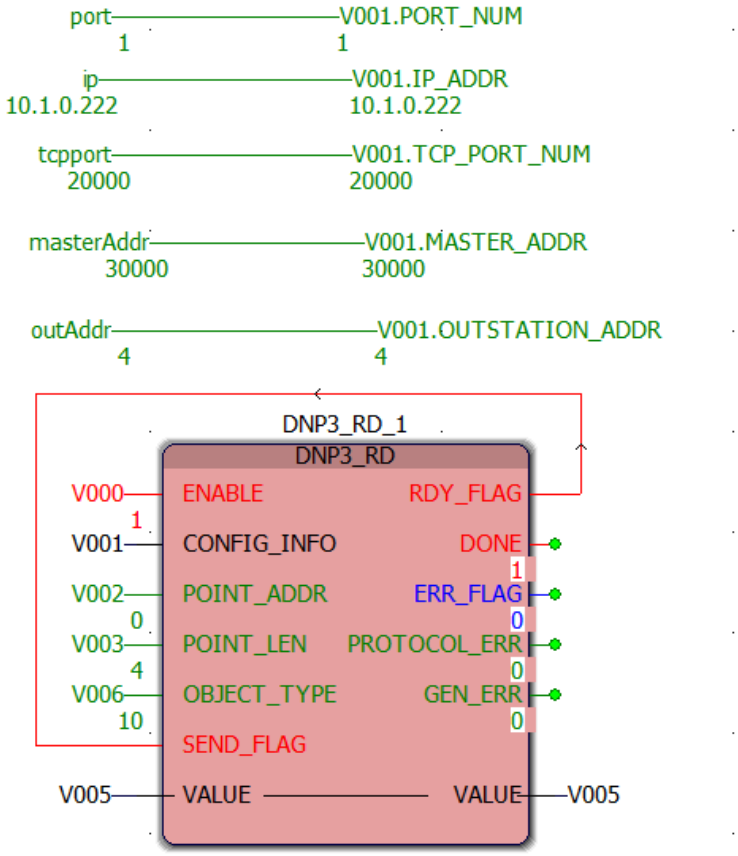
c. In the pop-up window of **Task settings**, configure the corresponding parameters.

d. Click **OK**.

3. Right-click the task you have inserted, and select **Insert** > **Program instance**.
4. Enter a name in the **Program instance** field. The program instance must not be named “RTU” or “GlobalVariable”.
5. Select the program you want to associate from the **Program type** drop-down list.
6. Right-click **Libraries** and select **Insert** > **Firmware Library**, select DNP3.FWL. Then click **Include**.
7. Under Logical POUs, double-click the code worksheet  of the program that you have inserted.
8. Drag the target function or function block of DNP3 from the Edit Wizard pane into the code worksheet, the function or function block is displayed. There are two function blocks available for DNP3 master programming. See Description of DNP3 Master Function Block for more information. For the following steps, the function block DNP3_RD is used as an example.
9. Double-click the pin-outs of the function or function block to assign variables. In the pop-up **Variable Properties** window, select the **Name**, **Data Type** and **Usage** from the drop-down list, and assign Initial value and I/O address. Then click **OK**.

   To assign initial values to CONFIG_INFO:

   CONFIG_INFO, a predefined data structure for DNP3 configuration information, is the crucial input for DNP3 master function blocks and contains key DNP3 communication parameters such as port number of the controller to be used, master address and outstation address, etc. This data structure is read-only and cannot be viewed and edited in ControlEdge Builder. See Description of CONFIG_INFO for more information.
10. After the basic programming steps as described, the workplace will appear as shown below.

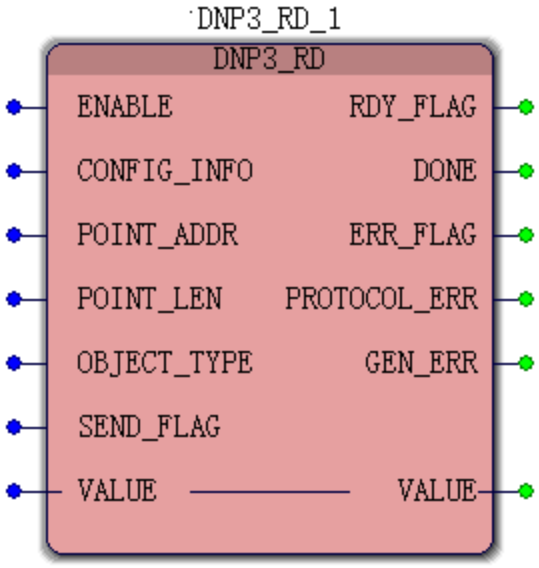


11. Click **Make** from the toolbar to compile the programs.
12. Click **Download** from the toolbar to download the compiled programs of DNP3 Master to the controller.

Description of DNP3 Master Function Block

There are 2 DNP3 Master function blocks available, Read Multiple Points and Write Multiple Points. With these function blocks, you can read and write Binary, Analog and String as per DNP3 protocol.

DNP3_RD



Description

It is used to read the following types of DNP3 points from outstation.

- Single-bit Binary Input
- Double-bit Binary Input
- Binary Output
- Analog Input
- Analog Output
- Counter
- Octet String

Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	DNP3_ CONFIG_ INFO	This is a structure provided by Honeywell. DNP3 Master related information is included. See Description of CONFIG_ INFO for more information.

Parameter	Data type	Description
POINT_ ADDR	UINT	The start point address you want to read from outstation.
POINT_LEN	UINT	The length of the points you want to read from outstation. The maximum length is 100 points.

OBJECT_ TYPE	USINT	DNP3 data object you want to read from outstation.
		This parameter can be set to the following values:
		kDnp3BinaryInput = 0;
		kDnp3BinaryOutputStatus = 1;
		kDnp3AnalogInput16 = 2;
		kDnp3AnalogInput16_NoFlag = 3;
		kDnp3AnalogOutput16Status = 4;
		kDnp3AnalogInput32 = 5;
		kDnp3AnalogInput32_NoFlag = 6;
		kDnp3AnalogOutput32Status = 7;
		kDnp3AnalogInputFloat = 8;
		kDnp3AnalogOutputFloatStatus = 9;
		kDnp3OctetStringRD = 10;
		kDnp3DoubleBitBinaryInput = 11;
		kDnp3Counter16 = 12;
		kDnp3Counter16_NoFlag = 13;
		kDnp3Counter32 = 14;
		kDnp3Counter32_NoFlag = 15;
		kDnp3FrozenCounter16 = 16;
		kDnp3FrozenCounter16_NoFlag = 17;
		kDnp3FrozenCounter32 = 18;
		kDnp3FrozenCounter32_NoFlag = 19;

Parameter	Data type	Description
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks will send the request. RDY_FLAG is TRUE means last communication is finished. Before last communication is finished, even if SEND_ FLAG is true the request won't be sent.

Output

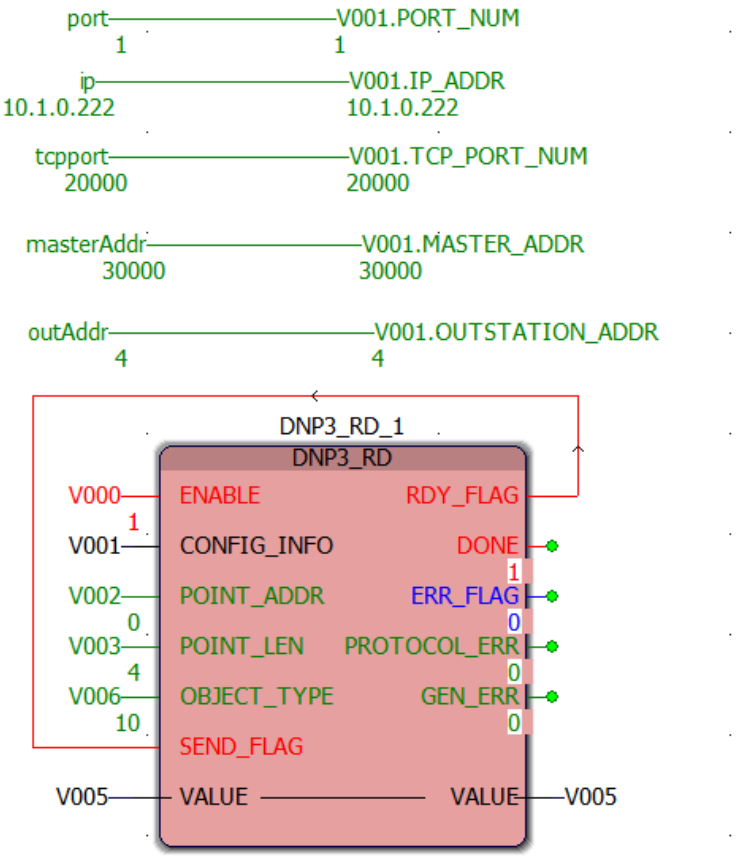
Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ ERR	USINT	Error numbers defined by DNP3 Master protocol. See DNP3 Master Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Input and Output

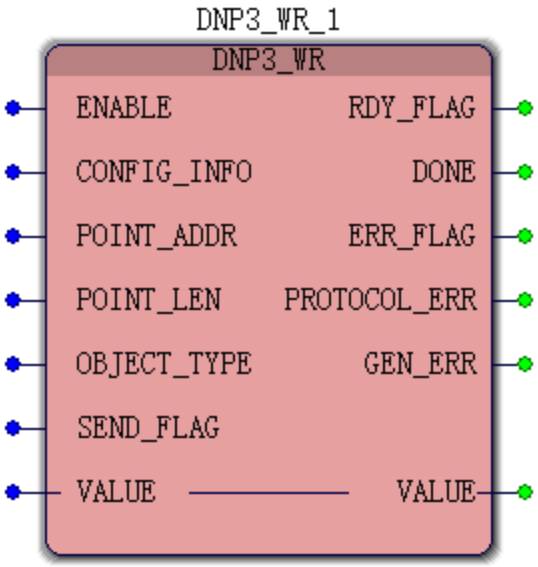
Parameter	Data type	Description
VALUE	DNP3_ DATA	Buffer for the data to be read (for read-output parameter)

Parameter	Data type	Description
		Buffer size = POINT_LEN*size of (data type) , maximum 512 bytes for this buffer. See the follow size of each data type: Dnp3BinaryInput (0) 1 byte Dnp3BinaryOutputStatus (1) 1 byte Dnp3AnalogInput16 (2) 2 bytes Dnp3AnalogInput16_NoFlag (3) 2 bytes Dnp3AnalogOutput16Status (4) 2 bytes Dnp3AnalogInput32 (5) 4 bytes Dnp3AnalogInput32_NoFlag (6) 4 bytes Dnp3AnalogOutput32Status (7) 4 bytes Dnp3AnalogInputFloat (8) 4 bytes Dnp3AnalogOutputFloatStatus (9) 4 bytes Dnp3OctetString (10) 1 byte Dnp3DoubleBitBinaryInput (11) 1 byte Dnp3Counter16 (12) 2 bytes Dnp3Counter16_NoFlag (13) 2 bytes Dnp3Counter32 (14) 4 bytes Dnp3Counter32_NoFlag (15) 4 bytes Dnp3FrozenCounter16 (16) 2 bytes Dnp3FrozenCounter16_NoFlag (17) 2 bytes Dnp3FrozenCounter32 (18) 4 bytes Dnp3FrozenCounter32_NoFlag (19) 4 bytes

Example


port
1
V001.PORT_NUM
1
ip
10.1.0.222
V001.IP_ADDR
10.1.0.222
tcpport
20000
V001.TCP_PORT_NUM
20000
masterAddr
30000
V001.MASTER_ADDR
30000
outAddr
4
V001.OUTSTATION_ADDR
4
DNP3_RD_1
DNP3_RD
V000
1
ENABLE
RDY_FLAG
V001
CONFIG_INFO
DONE
1
V002
0
POINT_ADDR
ERR_FLAG
0
V003
4
POINT_LEN
PROTOCOL_ERR
0
V006
10
OBJECT_TYPE
GEN_ERR
0
SEND_FLAG
V005
VALUE
VALUE
V005

DNP3_WR



Description

It is used to write the following types of DNP3 points from outstation.

- Single-bit Binary Input
- Double-bit Binary Input
- Binary Output
- Analog Input
- Analog Output
- Counter
- Octet String

Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	DNP3_ CONFIG_ INFO	This is a structure provided by Honeywell. DNP3 Master related information is included. See Description of CONFIG_ INFO for more information.

Parameter	Data type	Description
POINT_ ADDR	UINT	The start point address you want to write to outstation.
POINT_LEN	UINT	The length of the points you want to write to outstation. The maximum length is 100 points. **NOTE:** The maximum number of objects allowed in a single control request on external outstation side must be considered. If the number on the outstation side is less than 100, the "POINT_LEN" cannot exceed the number of the outstation.

OBJECT_ TYPE	USINT	DNP3 data object you want to write to outstation.
		
		This parameter can be set to the following values:
		
		kDnp3OctetStringWR = 20;
		
		kDnp3CROB_SelOp = 21;
		
		kDnp3CROB_DirOp = 22;
		
		kDnp3CROB_DONA = 23;
		
		kDnp3AnalogOutput16_SelOp = 24;
		
		kDnp3AnalogOutput16_DirOp = 25;
		
		kDnp3AnalogOutput16_DONA = 26;
		
		kDnp3AnalogOutput32_SelOp = 27;
		
		kDnp3AnalogOutput32_DirOp = 28;
		
		kDnp3AnalogOutput32_DONA = 29;
		
		kDnp3AnalogOutputFloat_SelOp = 30;
		
		kDnp3AnalogOutputFloat_DirOp = 31;
		
		kDnp3AnalogOutputFloat_DONA = 32;

SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks will send the request. RDY_FLAG is TRUE means last communication is finished. Before last communication is finished, even if SEND_ FLAG is true the request won't be sent.

Output

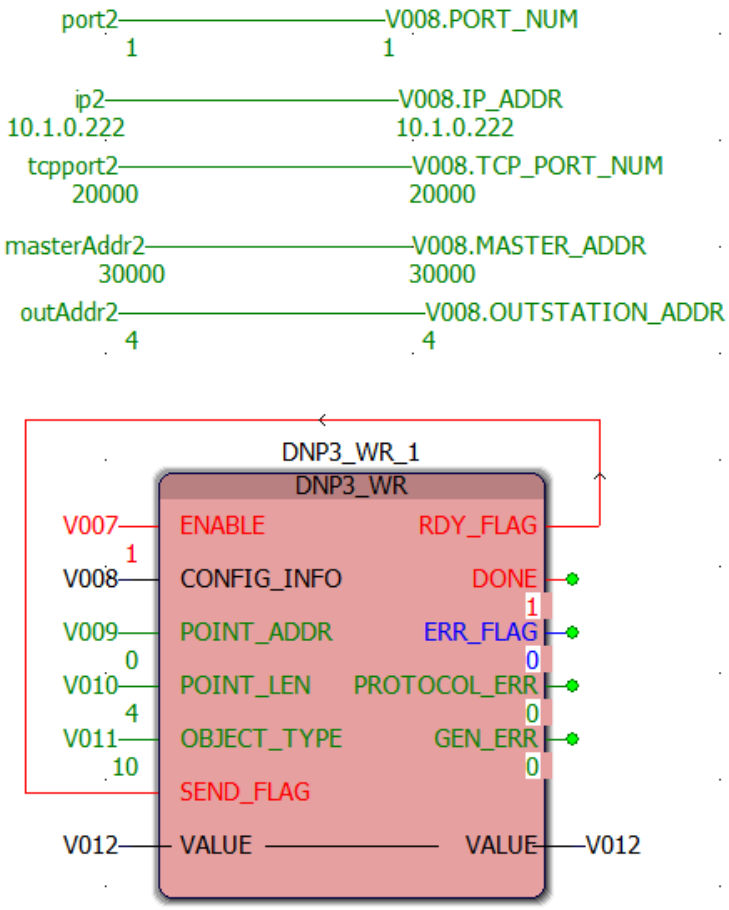
Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by DNP3 Master protocol. See DNP3 Master Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Input and Output

Parameter	Data type	Description
VALUE	DNP3_DATA	Buffer for the data to be read (for read-output parameter) Buffer size = POINT_LEN*size of (data type) , maximum 512 bytes for this buffer. See the follow size of each data type: Dnp3OctetStringWR (20) 1 byte Dnp3CROB_SelOp (21) 1 byte

Parameter	Data type	Description
		Dnp3CROB_DirOp (22) 1 byte Dnp3CROB_DONA (23) 1 byte Dnp3AnalogOutput16_SelOp (24) 2 bytes Dnp3AnalogOutput16_DirOp (25) 2 bytes Dnp3AnalogOutput16_DONA (26) 2 bytes Dnp3AnalogOutput32_SelOp (27) 4 bytes Dnp3AnalogOutput32_DirOp (28) 4 bytes Dnp3AnalogOutput32_DONA (29) 4 bytes Dnp3AnalogOutputFloat_SelOp (30) 4 bytes Dnp3AnalogOutputFloat_DirOp (31) 4 bytes Dnp3AnalogOutputFloat_DONA (32) 4 bytes

Example



Description of CONFIG_INFO

The CONFIG_INFO pin defined in the function blocks is to input all the configuration information for the DNP3 Master.

- For Ethernet communication of ControlEdge 2020 controllers, the data structure is defined as:

```
TYPE
        DNP3_CONFIG_INFO:
        STRUCT
                PORT_NUM:       UDINT;
                TCP_PORT_NUM:   UDINT;
```

```
                MASTER_ADDR:    UDINT;
                OUTSTATION_ADDR:UDINT;
                IP_ADDR:        STRING;
            END_STRUCT;


            (* Array data type for data read/write *)
            DNP3_DATA: ARRAY[1..512] of BYTE;
    END_TYPE
```

See the following table for the parameter descriptions:

Parameter	Data type	Description
PORT_NUM	UDINT	The physical interface of Ethernet port: 1. Ethernet port 1 2. Ethernet port 2
TCP_PORT_NUM	UDINT	TCP/IP port number of the DNP3 Master device
MASTER_ADDR	UDINT	The address of the DNP3 master
OUTSTATION_ADDR	UDINT	The address of the DNP3 outstation
IP_ADDR	STRING	The IP address of the DNP3 outstation device. Example: '192.168.0.100'

Description of Input and Output Data Type

See the following datatype of parameter Value for details:

DNP3_DATA

TYPE (* Array data type for data read/write *)

DNP3_DATA: ARRAY[1..512] of BYTE;

END_TYPE

DNP3 Master Protocol Error Codes

Refer to the following table for DNP3 Master Protocol Error Codes:

Error Code	Item	Description
0	SUCCESS	This indicates the request has completed successfully.
1	INTERMEDIATE	This indicates a response was received but the requested command is not yet complete. This could mean the response is part of a multi-fragment response and did not have the FINAL bit set. Or this could be a request such as a select operate that requires multiple requests and responses.
2	FAILURE	This indicates that the transmission of the request failed.
3	MISMATCH	The response to a select or an execute did not echo the request.
4	STATUSCODE	The response to a select or an execute echoed the request, except the status code was different indicating a failure.
5	IIN	The response to the request had IIN bits set indicating the command failed.
6	TIMEOUT	This indicates that the request has timed out. This could either be an incremental timeout indicating we received no link layer frame from the device in the specified time, or an application response timeout indicating this particular request did not complete in the specified time.
7	CANCELED	This indicates either that the user asked that the request be canceled by calling dnpchnl_cancel Fragment or that a second duplicate request has been made and therefore this first one is canceled.

CHAPTER

6 ENRON MODBUS SLAVE CONFIGURATION

Configuring an Enron Modbus Slave

To set a controller as an Enron Modbus Slave:

1. From the Home Page, click **Configure Ethernet Ports** and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the details in the **IP Address**, **Subnet Mask** and **Gateway** fields.
3. Under **Protocol Binding**, select **Enron Modbus Slave** to bind it to the Ethernet port.
4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.
5. Click **Configure Protocols** > **Enron Modbus Slave**, select the target Ethernet port you want to bind.
6. Select **Slave ID**. For Ethernet ports, the **Port** number must be configured.

   The port configured for Enron Modbus Slave cannot be the same port as that configured for Modbus Slave.
7. Click **Save**.
8. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
9. Click **Download** from the Home Page to load the configuration of the Modbus Slave to the controller.

CHAPTER

7 MODBUS SLAVE CONFIGURATION

Configuring a Modbus Slave

This section introduces how to set a controller as a Modbus TCP Slave or Modbus Serial Slave.

1. From the Home Page, click **Configure Ethernet Ports** to select an Ethernet port, or click **Configure Serial Ports** to select a serial port.
2. Configure corresponding parameters for the Ethernet or serial port.
3. Under **Protocol Binding**:
   - Select **Modbus Slave** for an Ethernet port.
   - Select **Modbus RTU Slave** or **Modbus ASCII Slave** for a serial port.
4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.
5. Click **Configure Protocols** > **Modbus Slave**, select the target Ethernet or serial port you want to bind.
6. Select **Slave ID**.
   - For Ethernet ports, the range is from 0 to 255
   - For Serial ports, the range is from 1 to 247
7. For Ethernet ports, configure the **TCP Port/UDP Port** number.
8. Select the required mapping table from the **Mapping** drop-down list.

   If the list is empty, you should add a mapping table first. See "Adding a Modbus Slave mapping table" in the *ControlEdge Builder User's Guide*.

   The same mapping table may be selected for use on multiple ports. For example, this could be used when a SCADA system communicates through 2 ports in for redundancy.
9. For Ethernet port, select **TCP** or **UDP** from drop-down list of **Type**.
10. For Ethernet ports, when the type is configured as TCP, set **Inactivity Timeout(s)** ranging from 20 to 86400.

> **NOTE:** The default value is 20. The configuration value must be greater than the scan rate of Modbus master.

11. Click **Save**.
12. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
13. Click **Download** from the Home Page to load the configuration of the Modbus Slave to the controller.

CHAPTER

8 MODBUS MASTER CONFIGURATION

Modbus TCP Master

Modbus TCP Master is used for communication between the controller and third-party Modbus slave devices over Ethernet. You need to bind the protocol to the Ethernet port of your controller and program the Modbus TCP Master for the project.

Configuring a Modbus TCP Master

A new project should be created and a controller should be added to the project opened in ControlEdge Builder. See "Creating a project" and "Connecting a controller" in *ControlEdge Builder User's Guide* for more details.

To set a controller as a Modbus TCP Master:

1. From the Home Page, click **Configure Ethernet Ports** and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the details in the **IP Address**, **Subnet Mask** and **Gateway** fields.
3. Under **Protocol Binding**, select **Modbus TCP Master** to bind Modbus TCP Master to the Ethernet port.
4. Click **Save** to save the configuration, or click **Back** to return to the Home Page.
5. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in the *ControlEdge Builder User's Guide*.
6. Click **Download** from the Home Page to load the configuration of Modbus TCP Master to the controller.

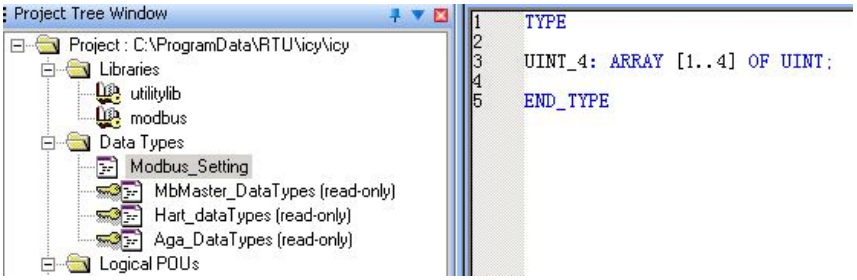
Programming a Modbus TCP Master

Follow the instructions below to program Modbus TCP Master for the project in **IEC Programming Workspace**.

1. Right-click **Logical POUs** and select **Insert** > **Program**. Then enter the name, and click **OK**. For the following steps, FBD language is used as an example.

2. Add a **Task** as follows:
    1. Under **Physical Hardware**, right-click **Task** and select **Insert** > **Task**.
    2. In the pop-up window, enter the name. Select the task type as **CYCLIC**, and click **OK**.
    3. In the pop-up window of **Task settings**, configure the corresponding parameters.
    4. Click **OK**.
3. Right-click the task you have inserted, and select **Insert** > **Program instance**.
4. Enter a name in the **Program instance** field.

    The program instance must not be named “RTU” or “GlobalVariable”.
5. Select the program you want to associate from the **Program type** drop-down list.
6. Right-click **Libraries** and select **Insert** > **Firmware Library**, select MODBUS.FWL. Then click **Include**.
7. Right-click **Data Types** and select **Insert** > **Datatypes**. In the pop-up window, enter the **Name** and click **OK**.
8. Double-click the data type you have inserted and define an array in worksheet shown as below as an example, then click **Save** button from the toolbar. Click **Make**.

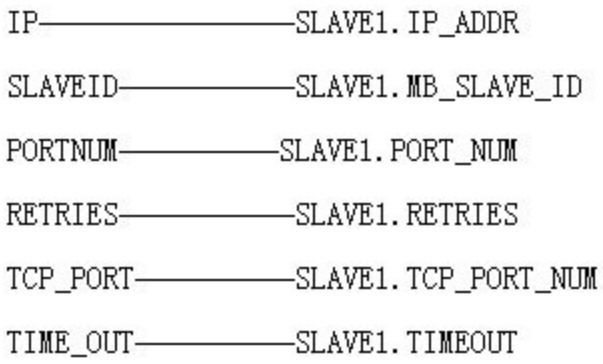


9. Under Logical POUs, double-click the code worksheet  of the program that you have inserted.
10. Drag the target function or function block of Modbus from the Edit Wizard pane into the code worksheet, the function or function block is displayed. There are twelve function blocks available for Modbus master programming. See Description of Modbus Function Block for more information. For the following steps, the function block MB_RD_MHR is used as an example.
11. Double-click the pin-outs of the function or function block to assign variables. In the pop-up **Variable Properties** window, select the **Name**, **Data Type** and **Usage** from the drop-down list, and

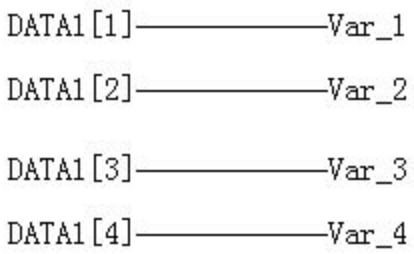
assign Initial value and I/O address. Then click **OK**. To assign initial values to CONFIG_INFO:

To assign initial values to CONFIG_INFO:

CONFIG_INFO, a predefined data structure for Modbus configuration information, is the crucial input for Modbus master function blocks and contains key Modbus communication parameters such as IP address of slave, slave ID, port number of the controller to be used, etc. This data structure is read-only and cannot be viewed and edited in ControlEdge Builder. See Description of CONFIG_INFO for more information. Slave1 is the variable name assigned by the user of CONFIG_INFO.

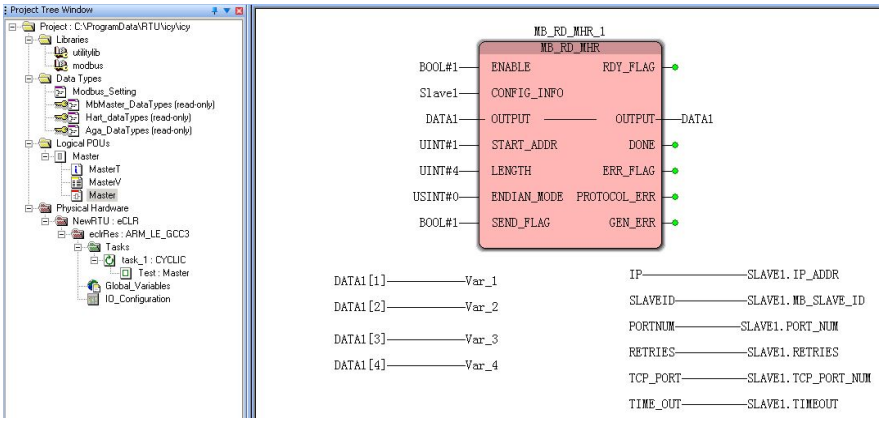


12. Assign the data returned by the function block to variables to monitor.



DATA1 is the variable name assigned by the user of OUTPUT pin of MB_RD_MHR and it is an array.

After the basic programming steps as described, the workplace will appear as shown below.



13. Click **Make** from the toolbar to compile the programs.
14. Click **Download** from the toolbar to download the compiled programs of Modbus TCP Master to the controller.

Modbus Serial Master

Modbus Serial Master is used for communication between the controller and third-party Modbus slave devices over a serial port. You need to bind the protocol to the serial port of your controller and program the Modbus Serial Master for the project.

Configuring a Modbus Serial Master

A new project is created and a controller is added to the project opened in RTU Builder. See "Creating a project" and "Connecting a controller" in *ControlEdge Builder User's Guide* for more details.

To set the controller as a Modbus Serial Master

1. From the Home Page, click **Configure Serial Ports** and select the target serial port to configure.
2. Under **General**,**Port Name** and **Port Type** are displayed automatically. Select appropriate values for **Baud Rate**, **Parity**, **Data Bits**, **Stop Bits**, **Flow Control** and **Force Online** if applicable. See the following tables for parameter descriptions.

   Table 8-1: Serial Port Parameters

Parameter	Description
Baud Rate	300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

Parameter	Description
	RS232 does not support 57600 and 115200.
Parity	None, ODD, EVEN
Data Bits	7, 8
Stop Bits	1, 2

For RS232-1 and RS232-2, there are two more options to configure: **Flow Control** and **Force Online**. See the following table for the parameter descriptions.

Table 8-2: RS232 Serial Port Parameters

Parameter	Description
Flow Control	Only for RS232-1 and RS232-2 • None • RTS-CTS
Force Online	Only for RS232-1 and RS232-2. Force Online is used to save energy when there is no device connected to the controller RS232 ports by disabling it. Select the desired option from the Force Online drop-down list: • Disable It is selected by default and the controller is on power saving mode. RS232 transmitter will detect the connection of external device. If external device is connected to the controller, the local transmitter will be enabled for communication. If there is no external device connected, the local transmitter will remain disabled to save energy. • Enable RS232 transmitter will not detect external device and if you force enable, more energy is consumed.

The following table describes four scenarios that will happen for **Force Online** option between the controller and the device it communicates.

Table 8-3: Force online scenarios between the controller and devices

Controller Force Online Option	Third-party Device Force Online Option	Communication
Enabled	Enabled	Normal
Disabled	Enabled	Normal, with energy saving on the controller
Enabled	Disabled	Normal, with energy saving on Device
Disabled	Disabled	It is forbidden. Both devices would consider there is no device connected to it and hence there is no communication between them.

3. Under **Protocol Binding**, select **Modbus RTU Master** or **Modbus ASCII Master** to bind Modbus Serial Master to the serial port. See the following table for parameter descriptions.

   Table 8-4: Parameter descriptions of Modbus RTU Master and Modbus ASCII Maste

Protocol	Description
Modbus RTU Master	The controller acts as the Modbus Master and used for communication between The controller and third-party Modbus Slave devices, for example I/O modules.
Modbus ASCII Master	The controller acts as the Modbus Master and used for communication between The controller and third-party Modbus Slave devices, for example: I/O modules.

4. Click **Save** to save the configuration, or click **Back** to return to the Home Page.
5. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.

6. Click **Download** from the Home Page to load the configuration of Modbus Serial Master to the controller.

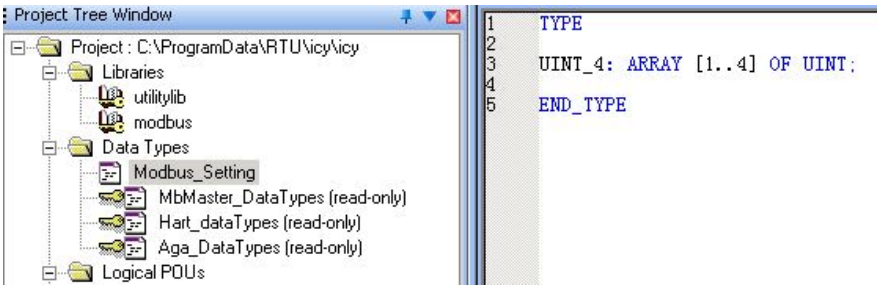
Programming a Modbus Serial Master

Follow the instructions below to program Modbus Serial Master for the project in **IEC Programming Workspace**.

1. Right-click **Logical POUs** and select **Insert** > **Program**. Then enter the name, and click **OK**. For the following steps, FBD language is used as an example.
2. Add a **Task** as follows:
   a. Under **Physical Hardware**, right-click **Task** and select **Insert** > **Task**.
   b. In the pop-up window, enter the name. Select the task type as **CYCLIC**, and click **OK**.
   c. In the pop-up window of **Task settings**, configure the corresponding parameters.
   d. Click **OK**.
3. Right-click the task you have inserted, and select **Insert** > **Program instance**.
4. Enter a name in the **Program instance** field.

   The program instance must not be named “RTU” or “GlobalVariable”.

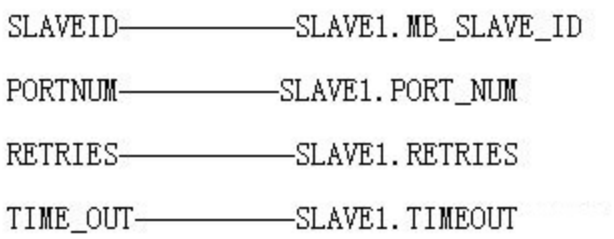
5. Select the program you want to associate from the **Program type** drop-down list.
6. Right-click **Libraries** and select **Insert** > **Firmware Library**, select MODBUS.FWL. Then click **Include**.
7. Right-click **Data Types** and select **Insert** > **Datatypes**. In the pop-up window, enter the **Name** and click **OK**.
8. Double-click the data type you have inserted and define an array in worksheet shown as below as an example, then click **Save** from the toolbar. Click **Make**.



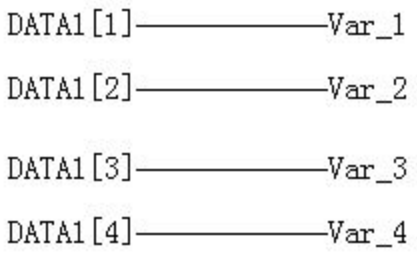
9. Under Logical POUs, double-click the code worksheet  of the program that you have inserted. The workspace appears.
10. Drag the target function or function block of modbus from the Edit Wizard pane into the workspace, the function or function block is displayed. There are twelve function blocks available for Modbus master programming. See Description of Modbus Function Block for more information. For the following steps, the function block MB_RD_MHR is taken as an example.
11. Double-click the pin-outs of the function or function block to assign variables. In the pop-up **Variable Properties** window, select the **Name**, **Data Type** and **Usage** from the drop-down list, and assign Initial value and I/O address. Then click **OK**.

    To assign initial values to CONFIG_INFO:

    CONFIG_INFO, a predefined data structure for Modbus configuration information, is the crucial input for Modbus master function blocks and contains key Modbus communication parameters such as IP address of slave, slave ID, port number of the controller to be used, etc. This data structure is read-only and cannot be viewed and edited in RTU Builder. See Description of CONFIG_INFO for more information. Slave1 is the variable name assigned by the user of CONFIG_INFO.

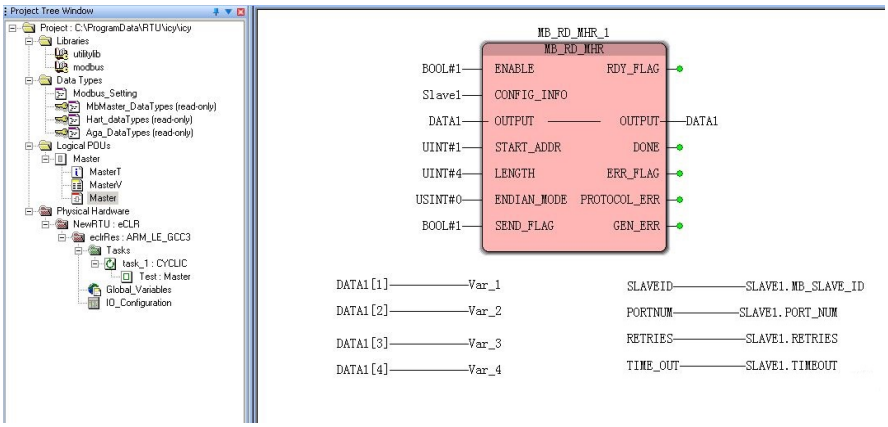


12. Assign the data returned by the function block to variables to monitor.



DATA1 is the variable name assigned by the user of OUTPUT pin of MB_RD_MHR and it is an array.

After the basic programming steps as described, the workplace will appear as shown below.

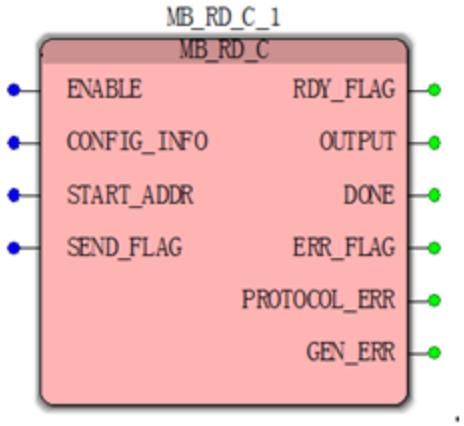


13. Click **Make** from the toolbar to compile the programs.
14. Click **Download** from the toolbar to download the compiled programs of Modbus Serial Master to the controller.

Description of Modbus Function Block

With these function blocks, you can read and write single coil, multiple coils, single discrete input, multiple discrete inputs, single input register, multiple input registers, single holding register, etc., as per Modbus protocol.

Read Single Coil



Description

It is used to read a single coil.

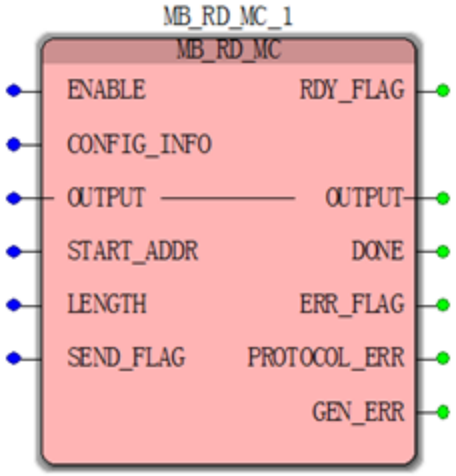
Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The first Modbus register address to read. Function code is not included in the address.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks will send the request. RDY_FLAG is TRUE means last communication is finished. Before the last communication is finished, even if the SEND_FLAG is true, the request won't be sent.

Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
OUTPUT	BOOL	Output: 1: true, 0: OFF
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Read Multiple Coils



Description

It is used to read multiple coils.

Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The first Modbus register address to read. Function code is not included in the address.
LENGTH	UINT	The number of registers to read, ranging from 1 to 2000.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks will send the request. RDY_FLAG is TRUE means the last communication finished. Before the last communication is finished, even if SEND_FLAG is true, the request won't be sent.

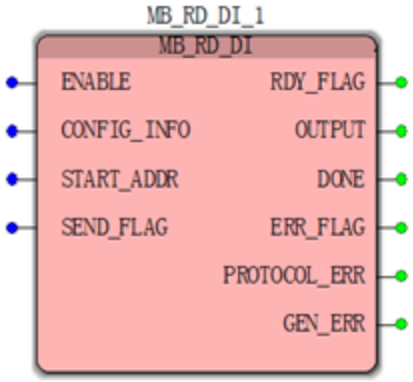
Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.

OUTPUT	Array of BOOL	User defined data type: array of bool. The size of the array should be equal to the number of the registers to read. Define a data type as shown below:
		`TYPE`
		`Variable Name: array[1..LENGTH] of BOOL;`
		`END_TYPE`

DONE	BOOL	Indicates that the response is received from a responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Read Single Discrete Input



Description

It is used to read single discrete input.

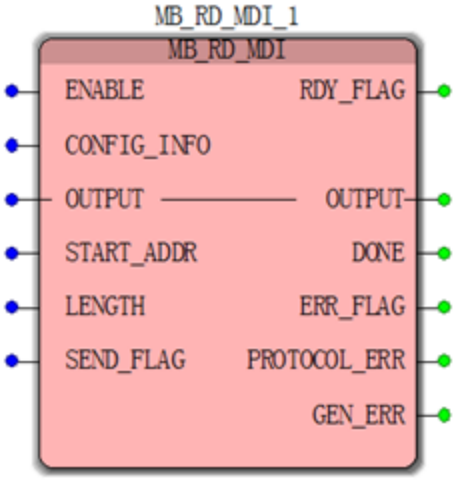
Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The first Modbus register address to read. Function code is not included in the address.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks will send the request. RDY_FLAG is TRUE means last communication is finished. Before last communication is finished, even if SEND_FLAG is true the request won't be sent.

Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
OUTPUT	Array of BOOL	User defined data type: array of BOOL. The size of the array should be equal to the number of the registers to read.
OUTPUT	BOOL	Output: 1: true, 0: OFF
DONE	BOOL	Indicates that the response is received from a responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Read Multiple Discrete Inputs



Description

It is used to read multiple discrete inputs.

Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The first Modbus register address to read. Function code is not included in the address.
LENGTH	UINT	The number of registers to read, ranging from 1 to 2000.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks will send the request. RDY_FLAG is TRUE means the last communication is finished. Before the last communication is finished, even if SEND_FLAG is true, the request won't be sent.

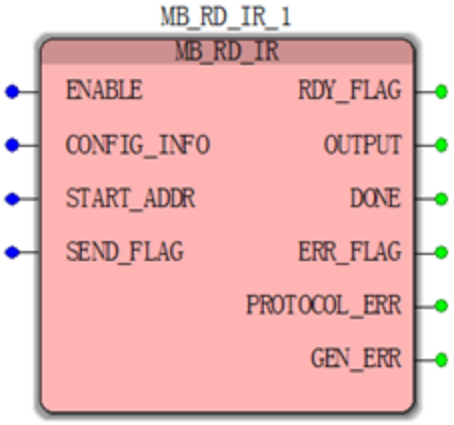
Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.

OUTPUT	Array of BOOL	User defined data type: array of bool. The size of the array should be equal to the number of the registers to read. Define a data type as shown here:
		`TYPE`
		`Variable Name: array[1..LENGTH] of BOOL;`
		`END_TYPE`

DONE	BOOL	Indicates that the response is received from a responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Read Single Input Register



Description

It is used to read single input register.

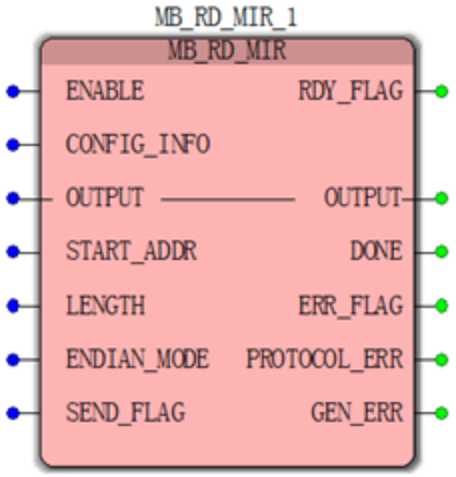
Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The Modbus register address to read. Function code is not included in the address.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks would send the request. RDY_FLAG is TRUE means the last communication is finished. Before last communication is finished, even if SEND_FLAG is true, the request won't be sent.

Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
OUTPUT	UINT	16bit Data read from the START_ADDR
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out(IPC timeout). 4: Invalid request

Read Multiple Input Registers



Description

It is used to read multiple input registers.

Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The first Modbus register address to read. Function code is not included in the address.
LENGTH	UINT	The number of registers to read, ranging from 1 to 125.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks would send the request. RDY_FLAG is TRUE means the last communication is finished. Before the last communication is finished, even if the SEND_FLAG is true, the request won't be sent.

Output

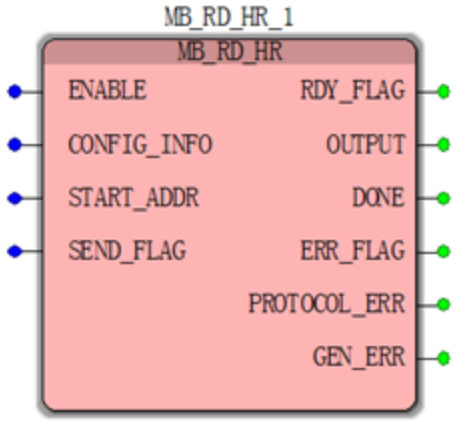
Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.

OUTPUT	INT, UINT, DINT, UDINT, LINT, REAL or LREAL;	User defined data type. The size of the array should be equal to the number of the registers to read multiplied by the register size.
		
		The end user should define a data type as shown here:
		
		`TYPE`
		`array[1..LENGTH] of`
		`INT/UINT/DINT/UDINT/LINT/REAL/LREAL;`
		`END_TYPE`
		
		The end user can read the data of a specific register by using the suffix.
		
		TIP: This block supports reading data from a Modbus responder configured with non-standard register sizes (For example: 32-bit or 64-bit registers).

ENDIAN_MODE	USINT	Endian mode is required for reading/writing 32bit and 64 bit variables. As Modbus always use big Endian to transceive data, there is no need to set the Endian mode for 16-bit data. 1: little Endian mode for 32 bit data 2: byte-swapped little Endian mode for 32 bit data 3: big Endian mode for 32 bit data 4: byte-swapped big Endian mode for 32 bit data 5: little Endian mode for 64 bit data 6: byte-swapped little Endian mode for 64 bit data 7: big Endian mode for 64 bit data 8: byte-swapped big Endian mode for 64 bit data See Endian Mode for more information.

Parameter	Data type	Description
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Read Single Holding Register



Description

It is used to read a single holding register.

Input

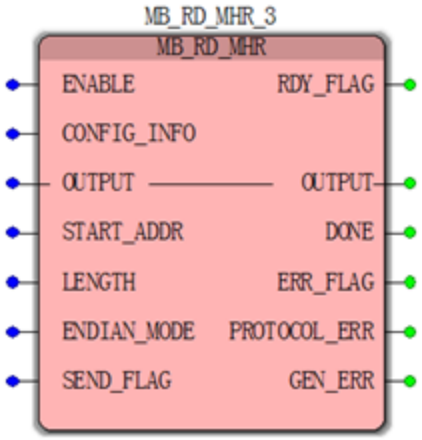
Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The Modbus register address to read. Function code is not included in the address.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks would send the request. RDY_FLAG is TRUE means the last communication is finished. Before the last communication is finished, even if the SEND_FLAG is true the request won't be sent.

Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
OUTPUT	UINT	16 bit data read from the START_ADDR
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set true if there is either a general error or a protocol error.
PROTOCOL_ ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded.

Parameter	Data type	Description
		1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Read Multiple Holding Registers



Description

It is used to read multiple holding registers.

Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_	UINT	The first Modbus register address to read. Function code is

Parameter	Data type	Description
ADDR		not included in the address.
LENGTH	UINT	The number of registers to read, ranging from 1 to 125.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks would send the request. RDY_FLAG is TRUE means the last communication is finished. Before the last communication is finished, even if the SEND_FLAG is true, the request won't be sent.

Output

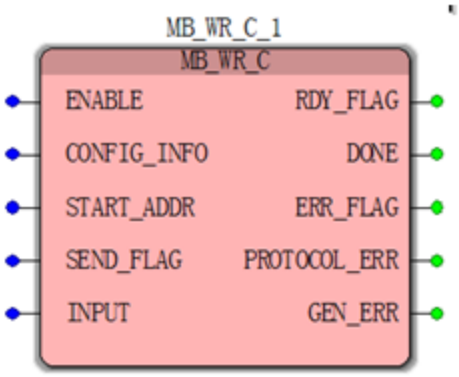
Parameter	Data type	Description
RDY_ FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.

OUTPUT	Array of INT, UINT, DINT, UDINT, LINT, REAL or LREAL;	User defined data type. The size of the array should be equal to the number of the registers to read multiplied by the register size.
		The end user should define a data type as shown here:
		`TYPE`
		`  Variable Name: array[1..LENGTH] of INT/UINT/DINT/UDINT/LINT/REAL/LREAL;`
		`END_TYPE`
		The end user can read the data of a specific register by using the suffix.
		TIP: This block supports reading data from a Modbus responder configured with non-standard register sizes (For example: 32-bit or 64-bit registers).

ENDIAN_ MODE	USINT	Endian mode is required for reading/writing 32bit and 64 bit variables. As Modbus always use big Endian to transceive data, there is no need to set the Endian mode for 16-bit data. 1: little Endian mode for 32 bit data 2: byte-swapped little Endian mode for 32 bit data

Parameter	Data type	Description
		3: big Endian mode for 32 bit data 4: byte-swapped big Endian mode for 32 bit data 5: little Endian mode for 64 bit data 6: byte-swapped little Endian mode for 64 bit data 7: big Endian mode for 64 bit data 8: byte-swapped big Endian mode for 64 bit data See Endian Mode for more information.
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Write Single Coil



Description

It is used to write a single coil.

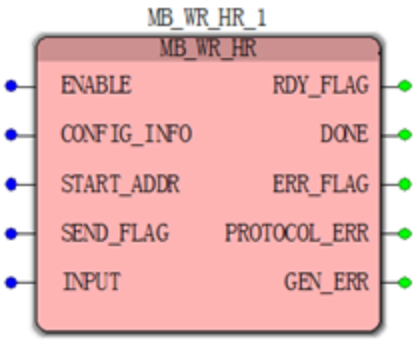
Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The Modbus register address to read. Function code is not included in the address.
SEND_ FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks would send the request. RDY_FLAG is TRUE means the last communication is finished. Before the last communication is finished, even if the SEND_FLAG is true, the request won't be sent.
INPUT	BOOL	1: ON 0: OFF

Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Write Single Holding Register



Description

It is used to write single holding register.

Input

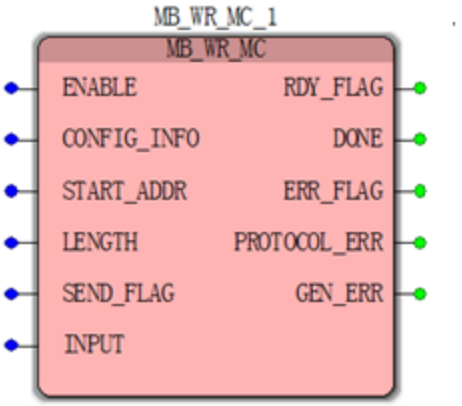
Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ADDR	UINT	The Modbus register address to read. Function code is not included in the address.
SEND_FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks would send the request. RDY_FLAG is TRUE means the last communication is finished. Before the last communication is finished, even if the SEND_FLAG is true, the request won't be sent.
INPUT	UINT	16 bit input data of START_ADDR register

Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded.

Parameter	Data type	Description
		1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Write Multiple Coils



Description

It is used to write multiple coils.

Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the FB is enabled and workable.
CONFIG_ INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ ADDR	UINT	The first Modbus register address to read. Function code is not included in the address.

Parameter	Data type	Description
LENGTH	UINT	The number of registers to write, ranging from 1 to 1968.
SEND_ FLAG	BOOL	If SEND_FLAG is TRUE and RDY_FLAG is true, function blocks would send the request. RDY_FLAG is TRUE means the last communication is finished. Before the last communication is finished, even if the SEND_FLAG is true, the request won't be sent.

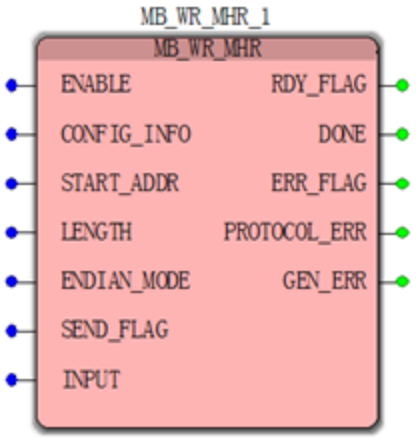
INPUT	Array of BOOL	User defined data type: array of bool. The size of the array should be equal to the number of the registers to read. The end user should define a data type as shown here:
		`TYPE`
		`Variable Name: array[1..LENGTH] of BOOL;`
		`END_TYPE`
		Use the suffix to set the status of a specific register.

Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. FB is ready for the next communication. False: command request is being sent or received.
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout

Parameter	Data type	Description
		3: Controller internal time out (IPC timeout). 4: Invalid request

Write Multiple Holding Registers



Description

It is used to write multiple holding registers.

Input

Parameter	Data type	Description
ENABLE	BOOL	Enable: If TRUE, the function block is enabled and workable.
CONFIG_INFO	User defined data type	This is a structure provided by Honeywell. Modbus related information is included. See Description of CONFIG_INFO for more information.
START_ADDR	UINT	The first Modbus register address to read. Function code is not included in the address.

Parameter	Data type	Description
LENGTH	UINT	The number of registers to write, ranging from 1 to 123.
SEND_FLAG	BOOL	If SEND_FLAG is true and RDY_FLAG is true, function blocks would send the request. RDY_FLAG is TRUE means the last communication is finished. Before the last communication is finished, even if the SEND_FLAG is true, the request won't be sent.

INPUT	Array of INT, UINT, DINT, UDINT, LINT, REAL, or LREAL	User defined data type. The size of the array depends on the number of the registers to write:
		Size of (array) * size of (element of array) / size of (UINT) = LENGTH.
		The end user should define a data type as shown here:
		`TYPE`
		`  Variable Name: array[1..LENGTH] of INT/UINT/DINT/UDINT/LINT/REAL/LREAL;`
		`END_TYPE`
		Use the suffix to read the data of a specific register.

Output

Parameter	Data type	Description
RDY_FLAG	BOOL	True: last communication is finished. The function block is ready for the next communication. False: command request is being sent or received.
ENDIAN_MODE	USINT	Endian mode is required for reading/writing 32bit and 64 bit variables. As Modbus always use big Endian to transceive data, there is no need to set the Endian mode for 16-bit data. 1: little Endian mode for 32 bit data 2: byte-swapped little Endian mode for 32 bit data 3: big Endian mode for 32 bit data 4: byte-swapped big Endian mode for 32 bit data

Parameter	Data type	Description
		5: little Endian mode for 64 bit data 6: byte-swapped little Endian mode for 64 bit data 7: big Endian mode for 64 bit data 8: byte-swapped big Endian mode for 64 bit data See Endian Mode for more information.
DONE	BOOL	Indicates that the response is received from responder device.
ERR_FLG	BOOL	Will be set to TRUE if there is either a general error or a protocol error.
PROTOCOL_ERR	USINT	Error numbers defined by Modbus protocol. See Modbus Protocol Error Codes for more information.
GEN_ERR	USINT	General error code: 0: Communication succeeded. 1: The input parameter is invalid. 2: Response timeout 3: Controller internal time out (IPC timeout). 4: Invalid request

Description of CONFIG_INFO

The CONFIG_INFO pin defined in the function blocks is to input all the configuration information for the Modbus master.

There are three types of communication between Modbus master and Modbus responder: serial communication of ControlEdge 2020 controllers using RS232 or RS485, Ethernet communication and serial communication of ControlEdge 900 Controllers. Accordingly three types of data structures are defined for CONFIG_INFO.

- For serial communication of ControlEdge 2020 controllers, the data structure is defined as:

```
TYPE
        MB_CONFIG_INFO_COM:
        STRUCT
```

```
                        MB_[1]RESPONDER_ID: USINT;
                        PORT_NUM:       USINT;
                        RETRIES:           USINT;
                        TIMEOUT:           UDINT;
        END_STRUCT;
END_TYPE
```

- For Ethernet communication, the data structure is defined as:

```
TYPE
        MB_CONFIG_INFO_ETH:
        STRUCT
                        MB_RESPONDER_ID:
USINT;
                        PORT_NUM:
USINT;
                        RETRIES:
USINT;
                        TIMEOUT:
UDINT;
                        TCP_PORT_NUM:       UINT;
                        IP_ADDR:
STRING;
        END_STRUCT;
END_TYPE
```

- For serial communication of ControlEdge 900 Controllers, the data structure is defined as:

```
TYPE
        MB_CONFIG_INFO_ECOM:
        STRUCT
                        MB_RESPONDER_ID: USINT;
                        PORT_NUM:          USINT;
                        RETRIES:           USINT;
                        TIMEOUT:           UDINT;
                        RACK_NUM:          UDINT;
                        SLOT_NUM:          UDINT;
        END_STRUCT;
END_TYPE
```

See the following table for the parameter descriptions:

[1]Adaption of new inclusive terminologies.

Parameter	Data type	Description
MB_[1]RESPONDER_ID	USINT	Modbus responder ID: valid arrange: 1~247.
PORT_NUM	USINT	The physical interface of serial port: 1. RS232 port 1 2. RS232 port 2 3. RS485 port 1 4. RS485 port 2 5. reserved 6. reserved The physical interface of Ethernet port: 1. Ethernet port 1 2. Ethernet port 2 3. reserved 4. reserved
RETRIES	USINT	Retry times before it is failed.
TIMEOUT	UDINT	Timeout unit: millisecond. The minimal timeout is 500 ms. If the end-user gives a number less than 500, the FB would send the default timeout value instead.
TCP_PORT_NUM	UINT	TCP/IP port number of the Modbus responder device
IP_ADDR	STRING	The IP address of the Modbus responder device. Example: '192.168.0.100'
RACK_NUM	UDINT	The rack number of the serial port: • 0 for local CPM,

[1]Adaption of new inclusive terminologies.

Parameter	Data type	Description
		• 1 to 99 for remote EPM
SLOT_NUM	UDINT	The slot number of the serial port, 1 to 12 are available

Description of Input and Output Data Type

Modbus supports reading and writing multiple consecutive registers. In these cases, the input or output is defined as an array.

- For reading and writing coils and discrete inputs, array of BOOL is defined.

  Set or retrieve the data value by using the suffix. For example: there are 10 coils to read, the output array COIL_OUT can be defined as array [1...10] of BOOL, reading the status of the fifth register could be COIL_OUT [5].

- For reading and writing input registers and holding registers, multiple array types can be defined: INT, UINT, DINT, UDINT, REAL, LREAL or LINT.

  Set or retrieve the data value by using the suffix. For example: there are 3 LREAL variables , or in other words, 12 holding registers to read, the output array LREAL_OUT can be defined as array[1..3] of LREAL, reading the value of the second register could be LREAL_OUT[2]. In this case, the Endian mode is involved.

Modbus Protocol Error Codes

Refer to the following table for Modbus Protocol Error Codes:

Error Code	Item	Description
0	success	N/A
65	I/O error	The underlaying I/O system reported an error.
69	Connection broken	Signals that the TCP/IP connection is closed by the remote peer or broken.
129	checksum error	N/A

Error Code	Item	Description
130	invalid frame error	Signals that a received frame does not correspond either by structure or content to the specification or does not match a previously sent query frame. A poor data link typically causes this error.
131	Invalid reply error	Signals that a received reply does not correspond to the specification
132	reply timeout error	Signals that a fieldbus data transfer timed out. This can occur if the responder device does not reply in time or does not reply at all. A wrong unit address will also cause this error. On some occasions, this exception is also produced if the characters received don't constitute a complete frame.
133	send timeout error	Signals that a fieldbus data send timed out. This can only occur if the handshake lines are not properly set.
134	Invalid responder[1] ID	Signals that a fieldbus data is not for me.
161	illegal function response	Signals that an illegal Function exception response was received. This exception response is sent by a responder device instead of a normal response message if a master sent a Modbus function not supported by the responder device.
162	illegal address response	Signals that an illegal Data Address exception response was received. This exception response is sent by a responder device instead of a normal response message if a master queried an invalid or non-existing data address.
163	illegal value response	Signals that an illegal Value exception response was received. This exception response is sent by a responder device instead of a normal response message if a master sent a data value that is not an allowed value for the responder device.
164	failure response	Signals that a Responder Device Failure exception response (code 04) was received. This exception response is sent by a responder device instead of a normal response message if an unrecoverable error occurred while processing the requested action. This response is also sent if the request would generate a response whose size exceeds the allowable data size.

[1]Adaption of new inclusive terminologies.

Error Code	Item	Description
165	Acknowledge	Responder has accepted request and is processing it, but a long duration of time is required. This response is returned to prevent a timeout error from occurring in the master. Master can next issue a Poll Program Complete message to determine whether processing is completed.
166	Responder Device Busy	Responder is engaged in processing a long-duration command. Master should retry later.
167	Negative Acknowledge	Responder cannot perform the programming functions. Master should request diagnostic or error information from responder.
168	Memory Parity Error	Responder detected a parity error in memory. Master can retry the request, but service may be required on the responder device.
170	Gateway Path Unavailable	Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate and an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
171	Gateway Target Device Failed to Respond	Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually mean that the device is not present on the network.

Endian Mode

Modbus protocol supports 16bit data only. If there are 32bit or 64bit variables, 2 or 4 consecutive registers should be used to read the data value. In these cases, the Endian mode may be involved due to the different Endian modes in Modbus responder devices.

See the following table for the concept of Endian modes used in Modbus function blocks:

Endian mode	Description
Little endian	Lower registers contain lower bits and higher registers contain higher bits. The order is on a register basis. Inside each register, the more significant byte is always at the first place as defined by the Modbus protocol.

Endian mode	Description
Big endian	Lower registers contain higher bits and higher registers contain lower bits. The order is on a register basis. Inside each register, the more significant byte is always at the first place as defined by the Modbus protocol.
Byte-swapped	The two bytes inside each register would be swapped.

See the following table for the valid Endian modes:

Valid Endian mode	Description
1	little Endian mode for 32 bit data
2	byte-swapped little Endian mode for 32 bit data
3	big Endian mode for 32 bit data
4	byte-swapped big Endian mode for 32 bit data
5	little Endian mode for 64 bit data
6	byte-swapped little Endian mode for 64 bit data
7	big Endian mode for 64 bit data
8	byte-swapped big Endian mode for 64 bit data

CHAPTER

9 OPC UA CONFIGURATION

Introduction

OPC is the interoperability standard for the secure and reliable exchange of data in the industrial automation space and in other industries. It is a platform independent and ensures the seamless flow of information among devices from multiple vendors.

OPC UA released in 2006 is a platform independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework.

ControlEdge 900 controller supports OPC UA server and client which are built-in protocols in the controller, and it provides an IIoT-ready open platform that enables users to better leverage data across their assets. Benefits include:

- Smooth integration with a broad range of instruments, equipment and software from multiple vendors
- Flexible and scalable design due to interoperable multi-level and multi-platform open communication
- Direct access to cloud-based applications for visualization and analytics

Prerequisite skills

This guidance uses terminology and concepts defined in OPC UA, PLCOpen and IEC 61131-3 specifications. It is assumed that you have familiarity with these Industry Standards. The following table lists relevant OPC UA or PLCOpen terminology and concepts introduced in this guide.

	Reference	Description
[UA-3]	OPC Unified Architecture Specification Part 3: Address Space Model,	Defines the basic address model concepts including nodes, attributes, references, variables, data types and methods. Standard node classes are defined in this specification.

	Reference	Description
	Release 1.03	
[UA-4]	OPC Unified Architecture Specification Part 4: Services, Release 1.03	Abstract descriptions of OPC UA Services which are organized into Service Sets. Familiarization with the “Overview” content for the following Service Sets is recommended: SecureChannel, Session, Monitored Item, and Subscription.
[UA-5]	OPC Unified Architecture Specification Part 5: Information Model, Release 1.03	Building on the concepts introduced in Part 3, this specification defines the UA Information model which is the base for all OPC UA Information Models including PLCOpen. It defines entry points into the address space, Server Object, built-in object or data types, standard objects and their variables and standard references.
[UA-7]	OPC Unified Architecture Specification Part 7: Profiles, Release 1.03	Defines meaningful collections of features for compliance purposes. Familiarity with the terminology introduced in this specification including profile, facet, and compliance unit is recommended.
[UA-12]	OPC Unified Architecture Specification Part 12: Discovery and Global Services	Describes how UA products can be discovered and managed on a computer, network infrastructure, or enterprise-wide.
[DI]	OPC Unified Architecture for Devices Companion Specification	Defines an OPC UA Information Model associated with Devices. This information model provides a base for other OPC UA companion specifications including PLC Open.
[PLC]	PLCopen and OPC Foundation: OPC UA Information Model for IEC 61131-3,	Defines an OPC UA Information Model to represent the IEC 61131-3 architectural models. This specification is considered a companion specification to the set of OPC UA specifications. To understand the overall structure of the IEC 61131-3 Information model as it relates to DI and base OPC UA, see the figure labeled as “OPC UA IEC 61131-3 ObjectTypes Overview”.

	Reference	Description
	Release 1.00	
[IEC]	IEC 61131 Basics	Help content that is accessible from ControlEdge Builder. IEC 61131 compliant PLC elements are described in detail including those for data types.

OPC UA Security

Security Objectives

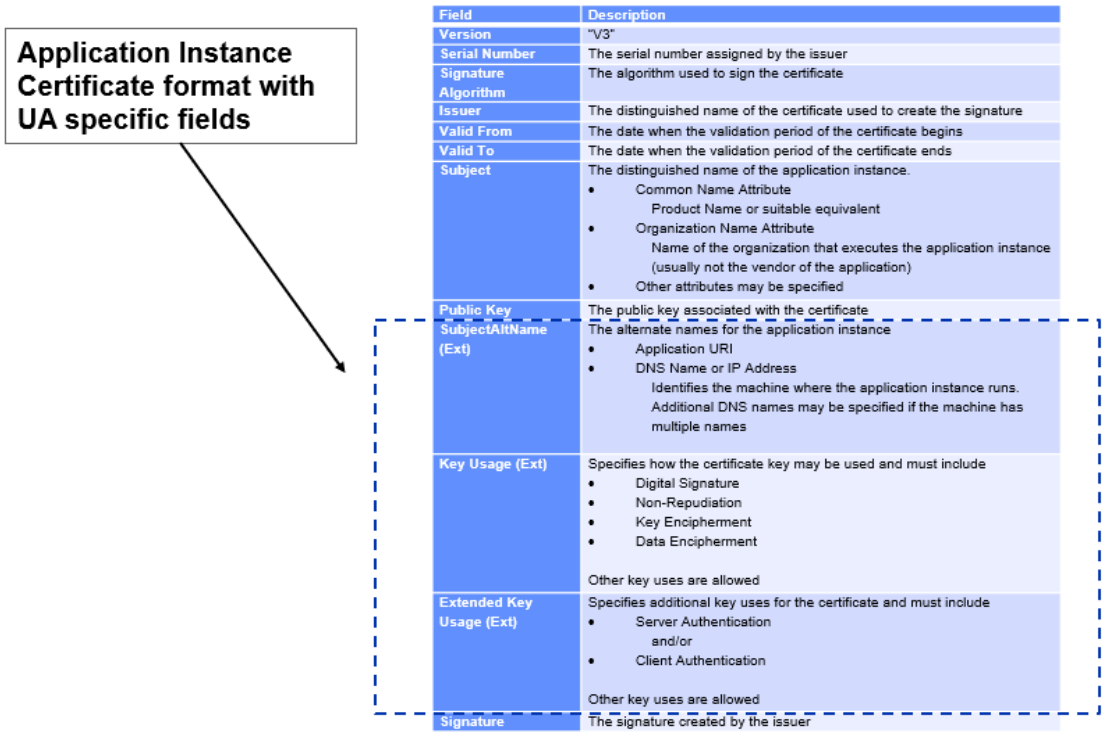
OPC UA defines several security objectives. In order to satisfy these security objectives, OPC UA requires that each instance of an OPC UA client and OPC UA server possess a unique x509 certificate known as an “application instance certificate”. OPC UA security objectives and the role of the application instance certificate in achieving these objectives are summarized in the table below.

Objective	Description
Authentication	The process of verifying the identity of an entity such as a client or server. In OPC UA, client and server, applications must exchange and validate each other’s certificate before a secure communication channel can be initiated.
Authorization	The right or permission granted to an entity to access a system resource. In OPC UA, client and server applications maintain “trust lists”. A client or server trust list identifies the set of applications which are authorized to access the resources of that client or server. When a client initiates communication with a server, certificates are first exchanged, then mutually authenticated then finally, compared to the set of authorized applications found in the local trust list. If the server’s trust list authorizes the client certificate and the client’s trust list authorizes the server certificate, then a secure communication channel can be established.
Confidentiality	Protection from data being read by unintended parties. After a secure communication channel has been established, client and server applications can utilize the exchanged certificate information to digitally encrypt message payloads exchanged between client and server. This ensures that malicious third parties are unable to read the content of the exchanged messages
Integrity	Assurance that information was not modified in transit. After a

Objective	Description
	secure communication channel has been established, client and server applications can utilize the exchanged certificate information to digitally sign message payloads exchanged between client and server. This ensures that malicious third parties are unable to successfully alter the content the exchanged messages.

Application Instance Certificates

Application instance certificates are uniquely assigned to individual client or server application instances. This means that different installations of one client or server application (e.g., same application installed on different host nodes) will have unique application instance certificates. Application instance certificate "uniqueness" is assured through the inclusion of special OPC UA extensions associated with the application instance certificate. Refer to the figure below. The extension "SubjectAltName" is crucial. The SubjectAltName extension must contain two pieces of information that ensure instance uniqueness. The Application URI component is a unique identifier assigned to the particular application instance and the DNS name or IP address component uniquely identifies the node which is hosting this application instance.



OPC UA Certificate Management

Due to the requirement that OPC UA application instance certificates hold instance unique information as described above, they can only be generated and assigned at client or server configuration time and only after the application has been installed. In most cases, OPC UA client and server applications are designed to self-generate a certificate, also known as a “self-signed” certificate. In many cases, self-signed certificates are sufficient to meet the security requirements of the deployment use case.

In more sophisticated deployment scenarios, a centralized certificate management service may exist. Such a service is generally known as a “certifying authority” or CA. The CA is responsible for generating and issuing application instance certificates for all of the OPC UA applications deployed within its scope of responsibility or authority. The CA is also responsible for managing the list of trusted applications within its scope which includes revoking certificates when an application is removed from the system. A comprehensive treatment of the roles and responsibilities generally associated with a CA is beyond the scope of this overview.

OPC UA Server Security

The ControlEdge OPC UA Server fully supports OPC UA Security by providing an application instance certificate for application authentication and implementing security policies defined by OPC UA.

Default Certificate

The UA Server on startup generates a self-signed application instance certificate. The certificate is unique to the ControlEdge PLC hosting the UA Server. This uniqueness is guaranteed by including the serial number of the ControlEdge PLC in the application URI. The application URI of the UA server is defined as follows:

`urn:[CPMSerialNumber]:Honeywell:ControlEdgePLC:UAServer`

In addition to the URI, the certificate also includes the IP Addresses assigned to both ETH1 and ETH2 thereby identifying the device where the UA Server is running. These elements are stored within the SubjectAltName extension of the application instance certificate. See Application Instance Certificates for more information.

Security Policies

Security policies define the security mechanisms used to secure the connection between the client and server. A security policy defines the algorithms for signing and encryption, the algorithm for key derivation and the key lengths used in the algorithms. ControlEdge PLC allows the user to configure the security policies that the UA Server will support. See Configure ControlEdge 900 controller OPC UA Server for more information.

User Authentication

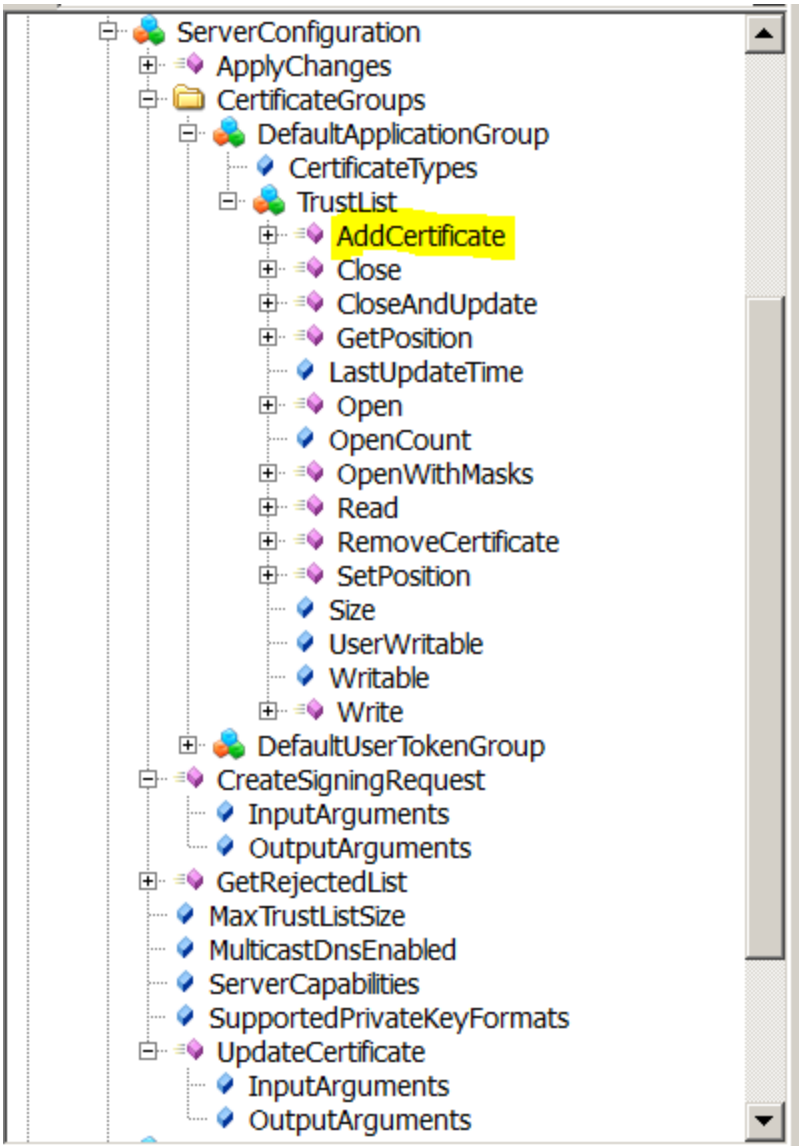
The UA Server supports user authentication/authorization by validating a username and password combination sent by the UA Client. The server will accept user credentials as provided by ControlEdge Builder namely Operator, Engineer or Administrator. UA Clients can select any one of these user names when connecting to the server.

User Authentication is configurable. The UA Server can also be configured to allow anonymous connections.

Trusting UA Client Applications

A secure connection between UA Client and the ControlEdge PLC UA Server is possible only when the UA server can trust and validate the client's application instance certificate. This requires that the client's application instance certificate be added to the server's trust list. To help with this process, the ControlEdge OPC UA Server supports the OPC UA "push management model" and exposes methods which can be invoked by authorized client applications in order to update the server's trust list when necessary. See [UA-12] for a detailed description of the push management model. Push model methods are provided by the server's ServerConfiguration object. This object provides a standard OPC UA interface for managing trust lists that allows external clients such as UAExpert to add client certificates to the ControlEdge PLC UA Server's trust list. In order to access the ServerConfiguration object, client applications must connect using a secure channel with encryption and supply Administrator credentials.

Shown below is the ServerConfiguration object and methods that it exposes:



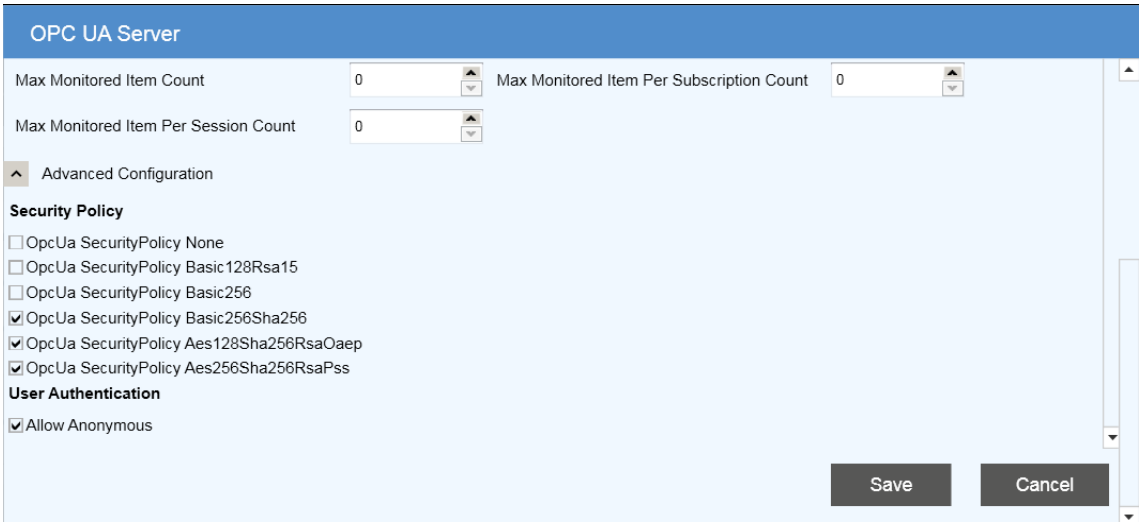
Provisioning mode

As part of the push model, the ControlEdge UA Server implements a “provisioning mode” which is a state during which the server will allow a secure client connection before any client certificates have been added to the server’s trust list. This is to allow a client application such as UA Expert to make the initial connection and add client certificates to the server trust list. Once one or more certificates have been added to the trust list the provisioning mode is turned off. It is therefore important to make sure that the certificate

of client application intended to be used for subsequent updates to the server trust list be added to the trust list while the UA Server is still in the provisioning mode. For details on using UAExpert to add certificates to the server trust list, see Add a Certificate to the Server Trust List for more information.

OPC UA Server Security Configuration

The “Advanced Configuration” tab allows an Administrator to configure security settings for the UA Server. The OPC UA Server can support one or more of the security policies listed below.



Security Policy Description

- OpcUa SecurityPolicy None. This policy is used for configurations with the lowest security needs. It results in a connection that is not secure.
- OpcUa SecurityPolicy Basic128Rsa15. This policy is used for configurations with medium security needs. It has been deprecated with the OPC UA Specification Version 1.04 and should be enabled only for backward compatibility
- OpcUa SecurityPolicy Basic256. This policy is used for configurations with medium security needs. It has been deprecated with the OPC UA Specification Version 1.04 and should be enabled only for backward compatibility
- OpcUa SecurityPolicy Basic256Sha256. This policy is used for configurations with high security needs.

- OpcUa SecurityPolicy Aes128Sha256RsaOaep. This policy is used for configurations with medium security needs.
- OpcUa SecurityPolicy Aes256-Sha256-RsaPss. This policy is used for configurations with high security needs.

User Authentication

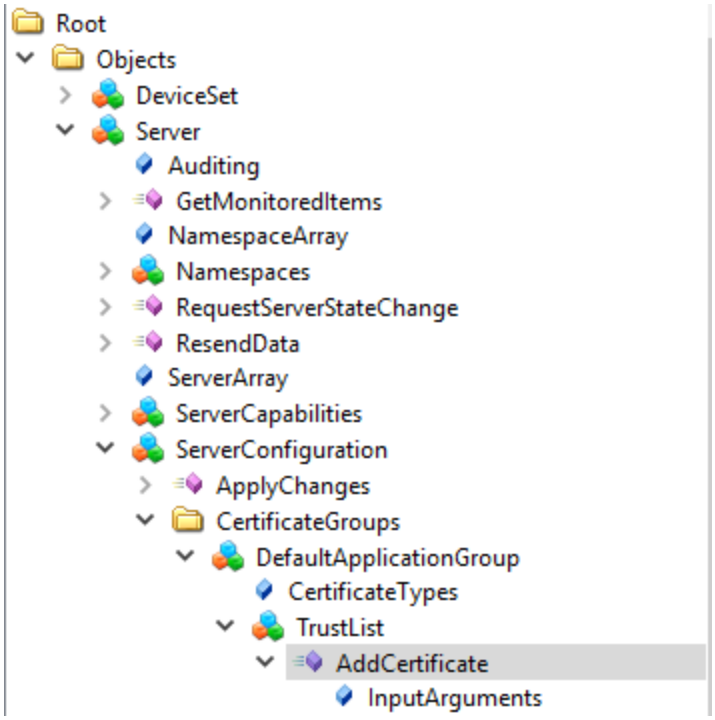
When “Allow Anonymous” is enabled from the ControlEdge Builder, UA Clients can access the UA Server without providing a username and password. When unchecked, UA Clients must provide credentials for one of the three supported user names, Administrator, Engineer or Operator.

Add a Certificate to the Server Trust List

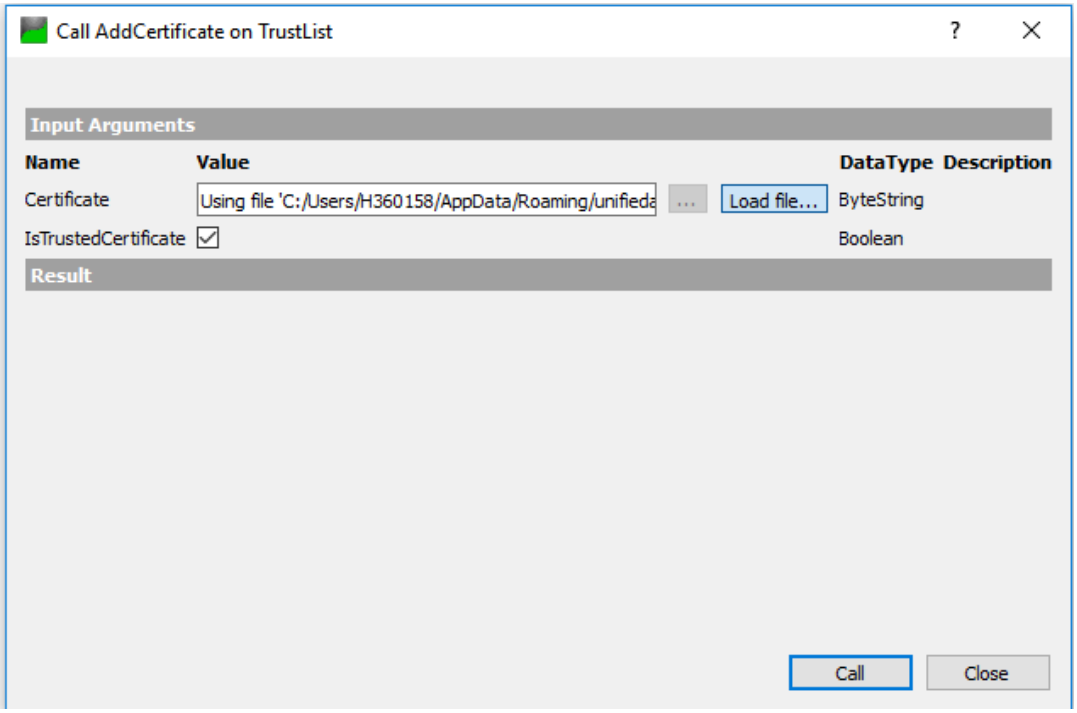
This example demonstrates how to use the UaExpert client to add a certificate to the server’s trust list.

1. Using the UaExpert client, connect to the server using security and provide Administrator credentials.
2. Browse the address space to find the method ***AddCertificate***.
   a. Path: `Root > Objects> Server > ServerConfiguration >CertificateGroups > DefaultApplicationGroup>`

`TrustList > AddCertificate`



3. Right click **AddCertificate** and select **Call**.
4. Click **Load file...**.
5. Load the UaExpert Client Certificate which can be found by going to **Settings** > **Manage Certificates** > **Open Certificate Location**.
   a. The certificate you are looking for should be in this path: `C:\Users\<user>\AppData\Roaming\unifiedautomation\uaexpert\PKI\own\certs\uaexpert.der`
6. Check **IsTrustedCertificate**.



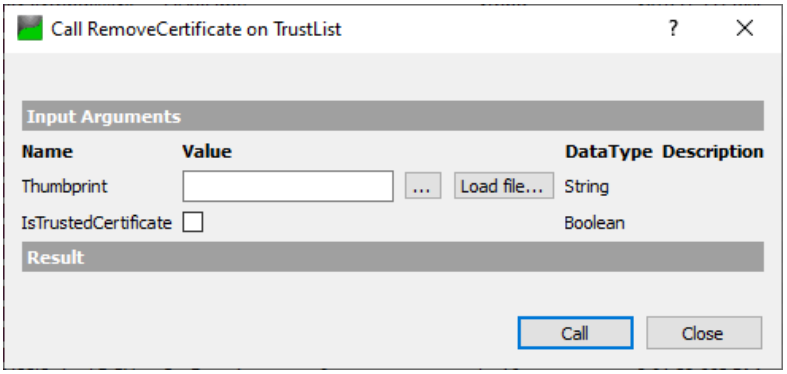
7. Click **Call**.
8. Restart the server.

The server should now have the UaExpert Client certificate in its trust list. Repeat this procedure as necessary for additional client certificates which need to be trusted by the server.

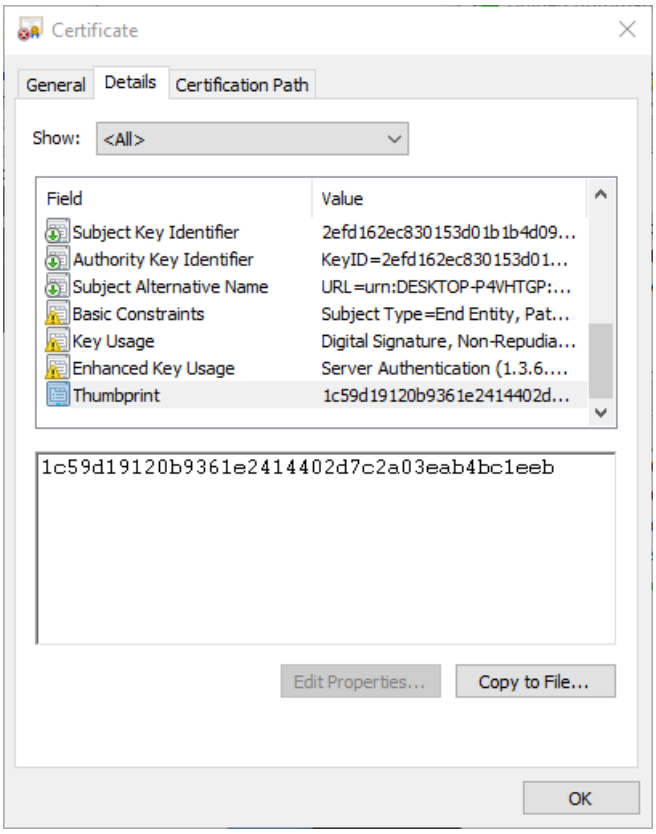
Remove a Certificate from the Server Trust List

To remove a certificate, use the method RemoveCertificate.

1. Path: `Root > Objects > Server > ServerConfiguration >CertificateGroups > DefaultApplicationGroup> TrustList > RemoveCertificate`
2. Right click **RemoveCertificate** and select **Call**.



3. Enter the Thumbprint of the certificate. Note that the Thumbprint entered should have no blanks and all the letters in the thumbprint must be uppercase. The thumbprint can be obtained by opening the certificate and looking in the details tab. Shown below is the thumbprint of uaexpert.der that was added in the AddCertificate example above.



4. Check **IsTrustedCertificate**.
5. Click **Call**.

OPC UA Client

The ControlEdge OPC UA Client fully supports OPC UA Security by providing an application instance certificate for application authentication and implementing security policies defined by OPC UA.

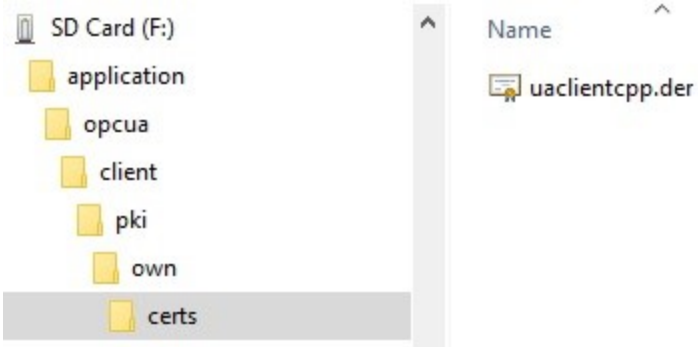
Default Certificate

The UA Client generates a self-signed application instance certificate when a project is downloaded to the controller using ControlEdge Builder. The certificate is unique to the ControlEdge PLC hosting the UA client. This uniqueness is guaranteed by including the serial number of the ControlEdge PLC in the application URI. The application URI of the UA server and is defined as follows:

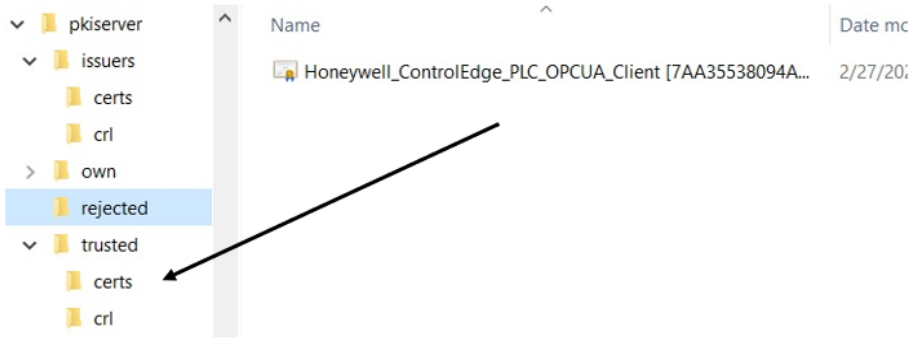
`urn:[CPMSerialNumber]:Honeywell:ControlEdgePLC:UAClient`

In addition to the URI, the certificate also includes the IP Addresses assigned to both ETH1 and ETH2 thereby identifying the device where the UA Client is running. These elements are stored within the SubjectAltName extension of the application instance certificate. See Application Instance Certificates for more information.

It will likely be necessary to obtain a copy of the ControlEdge PLC UA client application instance certificate so that it may be added to the trust list of a server to which the client will connect. Once the client application instance certificate is generated, a copy of the certificate is written to the memory card in the removable SD card slot if a card has been inserted. Any project download action will cause a copy of the certificate to be written to the SD card. Therefore, even after an initial download has been executed, an SD card can be inserted later and the project re-downloaded. The client certificate is written to the SD card at the location shown below.



In some deployments, access to a removable SD card may not be possible. When this is the case, simply initiate a secure connection to the target OPC UA server. If the server does not yet trust the client application instance certificate it will reject it and the UA_Connect function block will display a security error code (typically 0x80130000, “BadSecurityChecksFailed”). Next, examine the target server’s certificate storage and locate its “Rejected Certificates” folder.

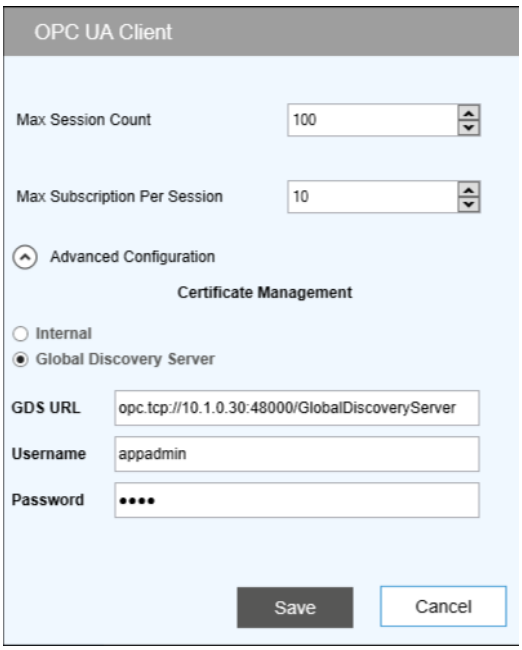


Here you will find a copy of the OPC UA client application instance certificate. Simply move the certificate it into the trusted folder.

OPC UA Global Discovery Services

The OPC UA client is capable of interacting with an OPC UA Global Discovery Server (GDS) in a manner similar to the “Push Model” of the OPC UA server as described earlier. In the case of the OPC UA client, this interaction is known as the “Pull Model”. With the Pull Model, the OPC UA client will contact the GDS in order to request or renew its application instance certificate and update its trust and revocation lists. The application instance certificate returned to the OPC UA client by the GDS is an issued certificate hence the GDS performs the role of a certifying authority or CA. Refer to OPC UA Certificate Management earlier in this section for a discussion on certifying authorities.

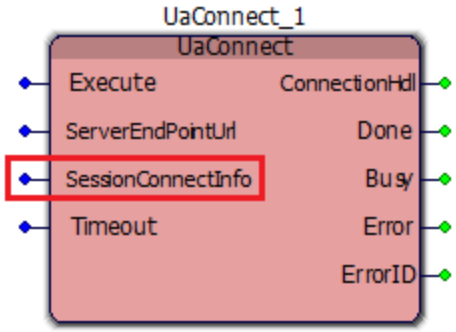
The OPC UA client must be configured to enable GDS interaction. Refer to the OPC UA Client configuration dialog within ControlEdge Builder and shown below.



There are two options available under the “Advanced Configuration” selection, “Internal” and “Global Discovery Server”. The Internal selection is default and configures the UA client to generate its own certificate. Selecting Global Discovery Server will enable the UA Client Pull Model for certificate management where it will contact the GDS and request an issued certificate. It is possible to switch between use of Global Discovery Server and Internal. However, once the new setting is selected, the project must be re-built and downloaded. Upon the next secure connection request via a UA_Connect function block (see below), either a new internal certificate will be generated (where Internal is selected), or the client will contact the GDS and request a new issued certificate. Note that in either case, upon first connect request following the switch between “Internal” and “Global Discovery Server” a new application instance certificate will be assigned to the OPC UA client and trust issues must be (re)considered with target UA servers. In general, connection to the Global Discovery Server will require authentication credentials in the form of GDS username/password. However, if the GDS does not require user authentication, the Username and Password fields may be left empty and the client will utilize an anonymous identity token when connecting to the GDS.

Securing a Connection

OPC UA client connections initiated from the ControlEdge PLC can be secured using the existing UaConnect function block and certain block input parameters. Refer to "*ControlEdge Builder Function and Function Block Configuration Reference Guide*", chapter 20 for complete detail on this function block and its inputs.



UASessionConnectInfo

UASessionConnectInfo	DataType	Description
SecurityMsgMode	UASecurityMsgMode	See UASecurityMsgMode section below.
SecurityPolicy	UASecurityPolicy	See UASecurityPolicy section below.
UserIdentityToken	UAUserIdentityToken	See UAUserIdentityToken section below.

Note that the UASessionConnectInfo structure has many other components in addition to the fields identified above. These three fields must be configured as detailed below in order to enable secure connections.

UASecurityMsgMode

Value	Name	Description
0	BestAvailable	Best available message security mode to the UA server. The client receives the available message security from the server and selects the best. This could also result in level "none security".
1	UASecurityMsgMode_	No security is applied. below.

Value	Name	Description
	None	
2	UASecurityMsgMode_Sign	All messages are signed but not encrypted.
3	UASecurityMsgMode_SignEncrypt	All messages are signed and encrypted.

UASecurityMsgMode is an integer value that configures the security level for exchanged messages on the connection. Options are “None”, that is, no security is applied, “Sign” which applies a digital signature to each message to ensure message integrity or “SignEncrypt” which means that all messages will additionally be encrypted, ensuring message confidentiality.

UASecurityPolicy

Value	Name	Description
0	UASecurityPolicy_BestAvailable	Provides the best available security connection to the UA server. The client receives the available policies from the server and selects the best. This can also result in level “none security”.
1	UASecurityPolicy_None	This policy is used for configurations with the lowest security needs. It results in a connection that is not secure.
2	UASecurityPolicy_Basic128Rsa15	This policy is used for configurations with medium security needs. It has been deprecated with the OPC UA Specification Version 1.04 and should be enabled only for backward compatibility.
3	UASecurityPolicy_Basic256	This policy is used for configurations with medium security needs. It has been deprecated with the OPC UA Specification Version 1.04 and should be enabled only for backward compatibility.
4	UASecurityPolicy_Basic256Sha256	This policy is used for configurations with high security needs.

UASecurityPolicy is an integer value that identifies the name for a set of security algorithms and cryptographic key lengths. The list above aligns with the set of security policies defined by the PLCOpen specification. Refer to [PLC] for additional information. Note that selecting '0' (UASecurityPolicy_BestAvailable) may result in a security policy which is not one of the above security policies (e.g., Aes128Sha256RsaOaep).

UAUserIdentityToken

<table>
<tr><th>UAUserIdentityToken</th><th>DataType</th><th>Description</th></tr>
<tr><td>UserIdentityTokenType</td><td>UAUserIdentityTokenType</td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>UAUITT_ Anonymous</td><td>See OPC UA Part 7 UserToken – Anonymous</td></tr>
<tr><td>1</td><td>UAUITT_ Username</td><td>See OPC UA Part 7 UserToken – User Name Password</td></tr>
<tr><td>2</td><td>UAUITT_ x509</td><td>See OPC UA Part 7 Chapter User Token – X509Certificate (Not supported)</td></tr>
<tr><td>3</td><td>UAUITT_ IssuedToken</td><td>See OPC UA Part 7 User Token – Issued Token (Not supported)</td></tr>
</table>
</td></tr>
<tr><td>TokenParam1</td><td>STRING</td><td>In case of TokenType "Anonymous" the Param1 will not be evaluated.

In case of TokenType "Username" the Param1 contains the user name.</td></tr>
<tr><td>TokenParam2</td><td>STRING</td><td>In case of TokenType "Anonymous" the Param2 will not be evaluated.</td></tr>
</table>

UAUserIdentityToken	DataType	Description
		In case of TokenType “Username” the Param2 contains the user password.

UAUserIdentityToken identifies the particular user associated with the connection. Where certificate exchange between client and server ensures mutual authentication, the user identity token ensures authorization. That is, OPC UA servers may restrict access to certain server resources. For example, the server might allow any user to read the value of any node in its address space but only certain users would be permitted write access. In this example, when the strategy of the program only requires reading values then a UAUserIdentityTokenType of UAUITT_Anonymous is appropriate. However, if the program strategy requires the ability to write to certain nodes then the UAUserIdentityTokenType would need to be set to UAUIT_Username and TokenParam1 and TokenParam2 configured with a username/password which has been granted write access by the server.

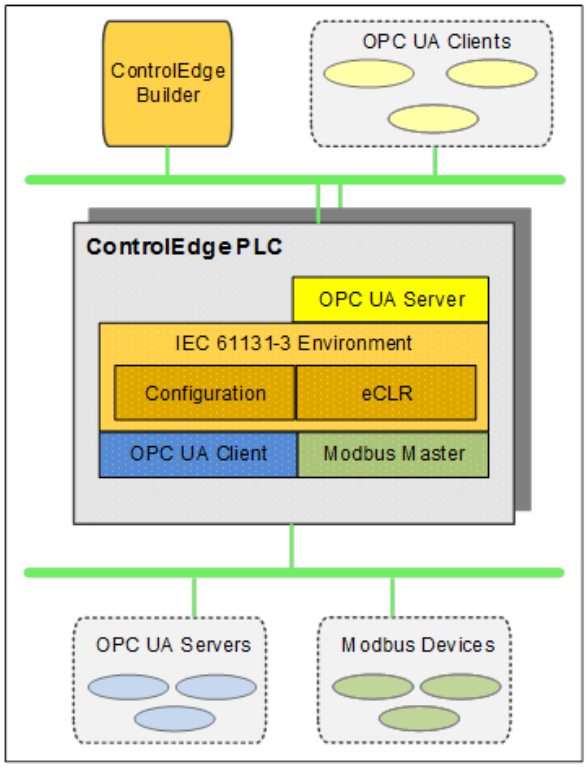
OPC UA Server

ControlEdge 900 controller OPC UA Server enables the native OPC UA client access to information on ControlEdge 900 controller.

System Architecture and Profiles

The figure below conceptually shows the deployment of the ControlEdge PLC OPC UA Server as an embedded OPC UA server. The same is true for the OPC UA Client and Modbus Master that are also shown in the figure below as examples of data sources. These examples are in addition to the local or remote I/O capabilities of ControlEdge 900 controller, all of which can be exposed by the ControlEdge 900 controller OPC UA server when the data sources are configured within the eCLR.

Although not shown below, it is possible to establish a peer to peer connection from the embedded OPC UA Client to the embedded OPC UA Server on a different ControlEdge 900 controller.



The ControlEdge 900 controller OPC Server is based on the Embedded UA Server profile defined in [OPC-7]. Refer to [PLC] for additional companion specification profile information.

Access Level

Currently, the ControlEdge 900 controller OPC UA server allows both read and write access of all exposed variables.

Security

Currently, the ControlEdge OPC UA Server is implemented with the lowest security level.

Redundancy

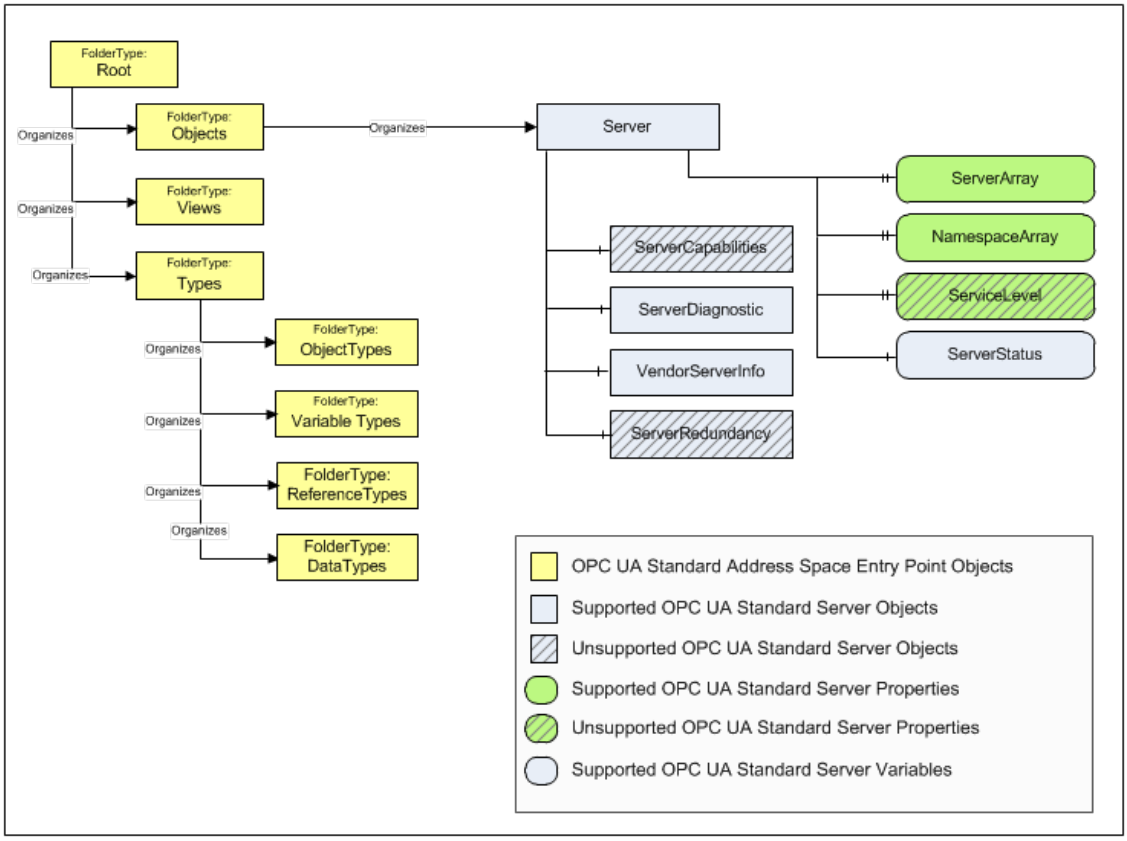
The ControlEdge OPC UA Server does not support UA redundancy as defined in [UA-4]. Furthermore, the ControlEdge OPC UA Server does not maintain any state data. Therefore, if an unexpected disconnection between the client and server occurs, it is the responsibility of the clients to re-establish connections (i.e. sessions). Even though none of OPC UA redundancy profiles are supported, the Control Edge OPC UA Server does participate in redundancy related usage scenarios supported in ControlEdge 900 controller.

Accessing the Server Object

The ControlEdge 900 controller OPC UA Server supports the standardized entry points into its address space.

- OPC UA clients can browse to the Server object by traversing the hierarchy starting at Root.
- Alternatively, OPC UA clients can use the Server object's well-known node id to directly access its objects, properties and variables.

Use the diagram legend to understand the objects, variables and properties that the ControlEdge 900 controller OPC UA Server supports.



Server Diagnostics

The Sever Diagnostic object shown above, represents pertinent diagnostic information related to the ControlEdge 900 controller OPC UA server itself. All mandatory sub-components and properties are supported [UA-5].

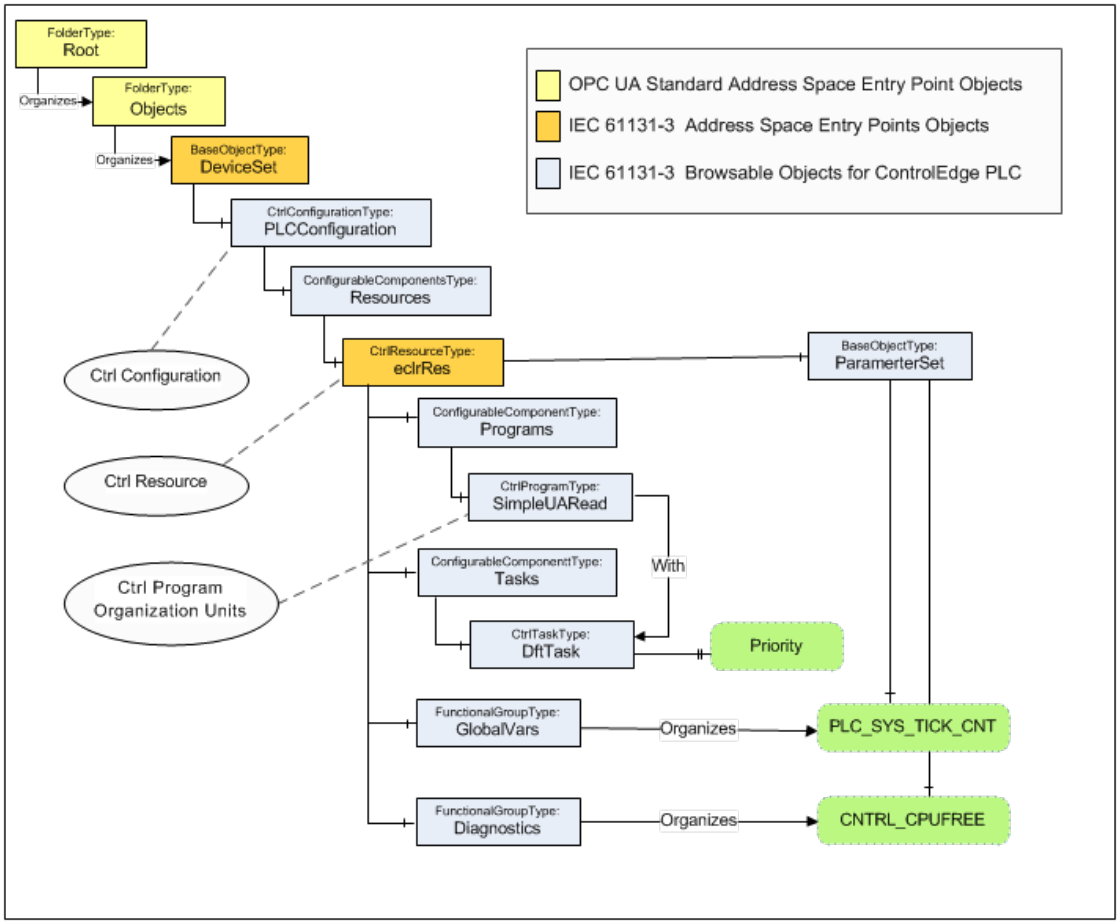
Accessing ControlEdge PLC data

Overview

DeviceSet is the entry point for OPC UA Clients that want to access data from ControlEdge 900 controller. Shown below is an example address space of the ControlEdge 900 controller OPC UA Server. It is based on Object Types definitions found in the base UA specification ([UA-5]) as well as those definitions found in companion specifications ([DI] and [PLC]). In this address space, there are three example objects:

- eclrRes is an example of a Ctrl Resource
- SimpleUARead is an example of a Ctrl Program Instance.
- DftTask is an example of a Ctrl Task.

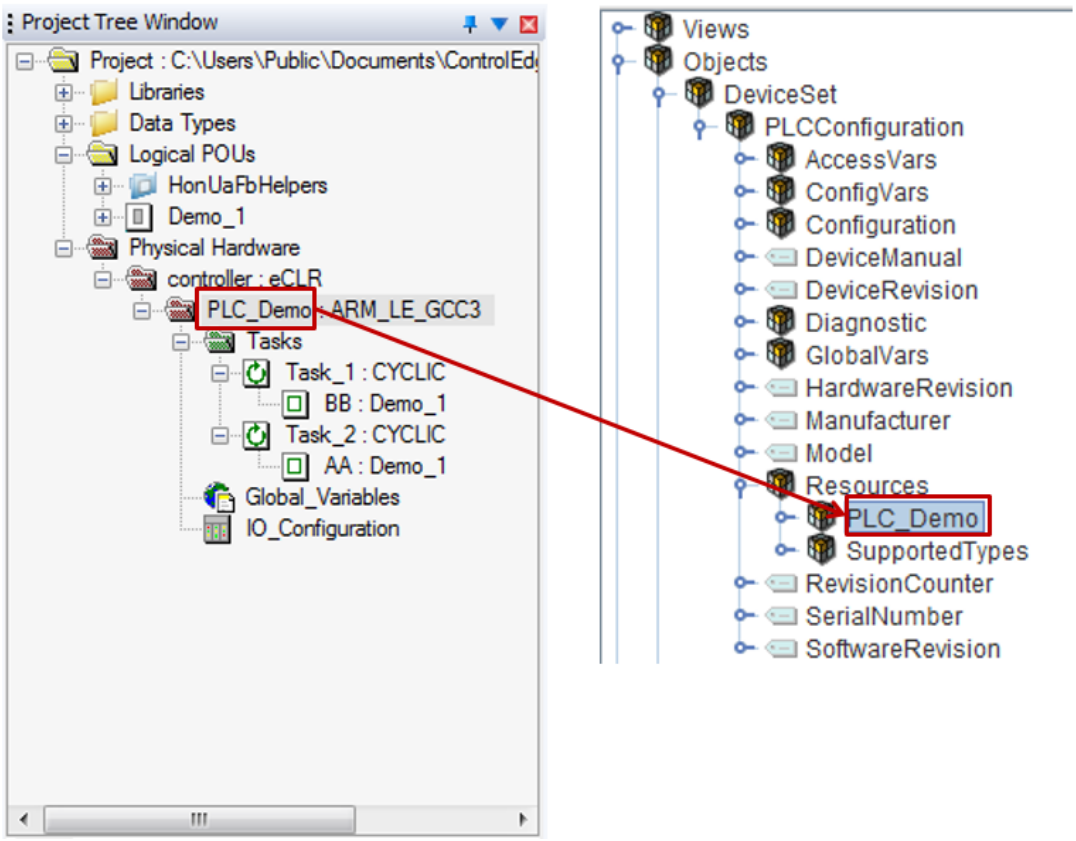
Figure 9-1: An example for address space of the ControlEdge 900 controller OPC UA Server



Ctrl Resources

All the data on ControlEdge 900 controller is accessible by browsing to the Object instance derived from CtrlResourceType. The Browsename of this Object instance is the name given to the resource that represents the controller itself.

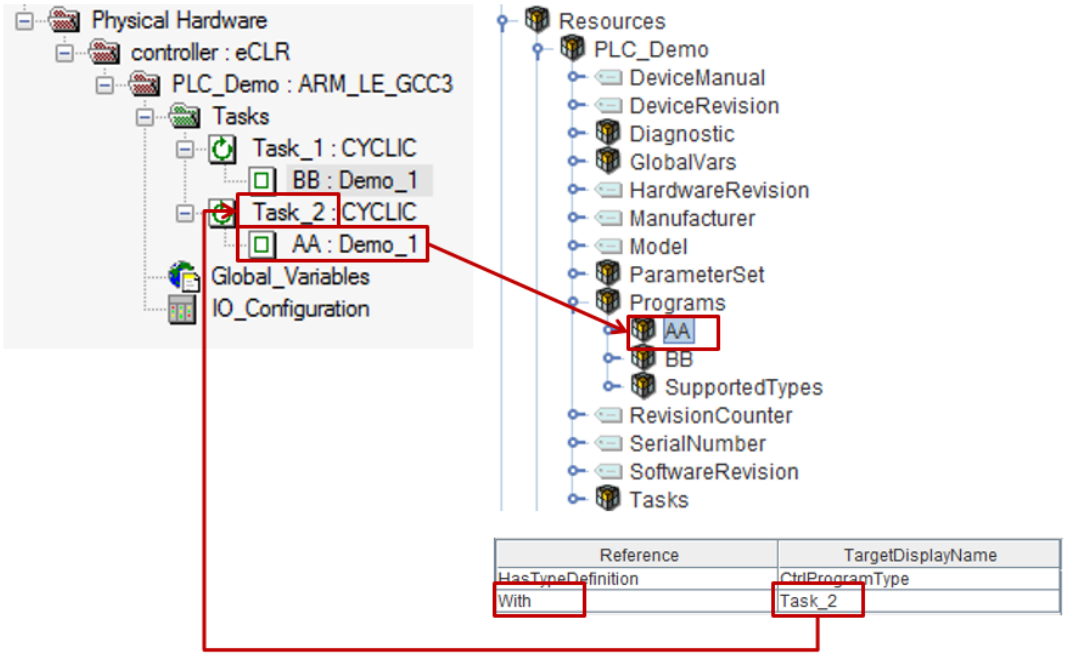
Shown below is the output of a 3rd party client connected to the ControlEdge 900 controller OPC UA Server. The objects listed below PLC_Demo represents the complete set of data associated with ControlEdge 900 controller including Programs, Tasks, Global variables and Diagnostics.



Ctrl Programs

In ControlEdge 900 controller, the instance of a POU assigned to a task is treated as a program instance. Note that same program instance in is able to be assigned to different task. They are treated as different program instance.

Shown below is the output of a 3rd party client connected to the ControlEdge 900 controller OPC UA Server, all Ctrl Program instances executing in ControlEdge 900 controller are located under Programs. For each Ctrl Program, the program variables and Function Block instances including their child function blocks and variables also appear in the address space.



The “With” reference is used to show the association between the program instance and the task that executes the program.

ControlEdge 900 controller Diagnostics

See Overview for more information. OPC UA clients have access to ControlEdge PLC diagnostic information exposed by the GlobalVars and Diagnostics folder objects.

Located under the GlobalVars folder are the ControlEdge PLC System Variables including PLC_SYS_TICK_CNT and PLC_MAX_ERRORS.

Located under the Diagnostics folder are the ControlEdge 900 controller diagnostics as viewed from the ControlEdge Configuration Workspace of ControlEdge Builder.

Program Variable NodeIds

OPC UA Clients can use NodeIds to read, write and monitor variables for data changes. The NodeIds for accessing IEC 61131-3 program elements are defined with string identifiers. The string identifiers embed the underlying name that the ControlEdge 900 controller OPC UA Server uses to access the ControlEdge 900 controller variables. See Global and Diagnostic Variables for more information.

See Global and Diagnostic Variables for more information.

See Program Variables for more information.

Global and Diagnostic Variables

For Global and Diagnostic variables, the Identifier element of the NodeId is defined as: Identifier = @GV. <Varname>

For example, the NodeId of the global variable ‘PLC_SYS_TICK_CNT’ is shown below.

Element	NodeId
NamespaceIndex	
IdentifierType	String
Identifier	@GV.PLC_SYS_TICK_CNT

> **TIP:** For ControlEdge 900 controller OPC UA client, function block “UaNamespaceGetIndex” is able to get the NamespaceIndex when NamespaceUri of ControlEdge 900 controller OPC Server is available. See Key Parameters to establish OPC UA communication for more information.

Program Variables

For Program local variables, the Identifier element of the Nodeid is defined as:

Identifier = <Program Instance Name>.<Varname>

For example, the NodeId of the local variable ‘Connect’ defined within the program instance ‘ReadWrite is shown below.

Element	NodeId
NamespaceIndex	

Element	NodeId
IdentifierType	String
Identifier	ReadWrite.Connect

For Function Block instance variables, the Identifier element of the NodeId is define as:

Identifier = <Program Instance Name>.<FunctionBlockInstance>.<Varname>

The example below shows the NodeId of 'ConnectHandle' which is a variable of the Function Block instance 'UA_Read_Write_1' in the program 'ReadWrite'

Element	NodeId
NamespaceIndex	
IdentifierType	String
Identifier	ReadWrite.UA_ReadWrite_1.ConnectionHandle

Data Types

Elementary types

The ControlEdge 900 controller OPC UA server maps all IEC 61131-3 elementary data types supported on the controller to an OPC UA built in data type. The table below shows how the elementary data types defined by IEC 61131 -3 map to OPC UA Built in data types.

IEC 61131-3 Elementary Data Types	OPC UA Built In Data Types
BOOL	Boolean
SINT	SByte
USINT	Byte
INT	Int16
UINT	UInt16
DINT	Int32
UDINT	Uint32
BYTE	Byte

IEC 61131-3 Elementary Data Types	OPC UA Built In Data Types
WORD	UInt16
DWORD	Uint32
REAL	Float
LREAL	Double
STRING	String
TIME	Double

> **TIP:** When writing to a variable, the ControlEdge OPC UA Server shall return a Bad_TypeMismatch error if the data type of the written value is not the same type or subtype of the variable's DataType.

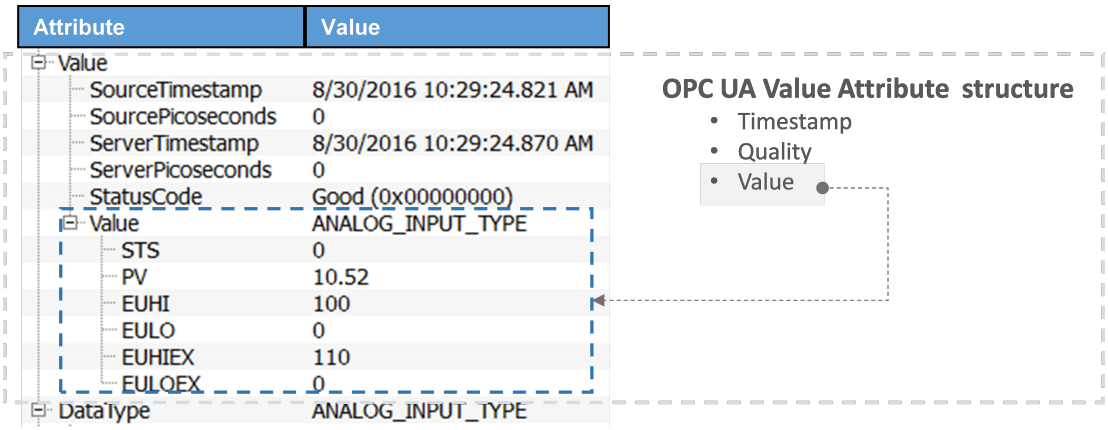
Structured Types

The IEC 61131-3 STRUCT declaration represents a structured data type as an aggregate data type. IEC 61131-3 structured data types defined in ControlEdge 900 controller are mapped by the ControlEdge OPC UA Server to OPC UA structured data types as defined in [PLC] section 5.2.3.4. Shown below is the definition of ANALOG_INPUT_TYPE as an IEC 61131-3 structured type defined in ControlEdge 900 controller.

```
TYPE
      ANALOG_INPUT_TYPE
      STRUCT
                  STS       :       USINT;
                  PV        :       REAL;
                  EUHI      :       REAL;
                  EULO      :       REAL;
                  EUHIEX    :       REAL;
                  EULOEX    :       REAL;
      END_STRUCT;
END_TYPE
```

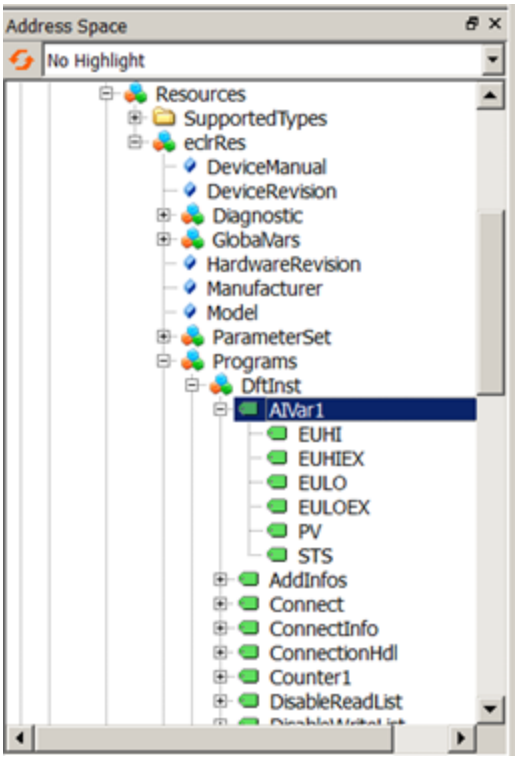
To illustrate how the ControlEdge 900 controller OPC UA Server maps ANALOG_INPUT_TYPE, we connect a 3rd party sample client to the ControlEdge 900 controller OPC UA Server.

Shown below is the partial Browse output for a variable.



Currently, the ControlEdge 900 controller OPC UA server implements approach "c" as described in the section entitled “Many Variables and / or structured DataTypes” of [UA-3]. This means that an OPC UA client can access the whole data structure as well as its individual elements.

Shown below is the AddressSpace showing the variable, AIVar1 and individual elements of the variable.



The following steps summarize how the ControlEdge 900 controller OPC UA Server exposes IEC 61131-3 Structured Data Types such as ANALOG_INPUT_TYPE to OPC UA Clients:

1. Creates an OPC UA Structured DataType with the same elements as the IEC 61131-3 STRUCT.
2. Creates an OPC UA Complex Variable with the DataType created in step 1.
3. Creates several simple Variables using simple DataTypes to reflect the elements in the IEC 61131-3 STRUCT and exposes them as variables of the Complex Variable created in step 2.
4. Adds the Variable created in step 2 to the AddressSpace to make the data available to the OPC UA Client.

Additionally, the Datatype, which in our example is ANALOG_INPUT_TYPE, is added to the DataTypeDictionary. The DataTypeDictionary also contains all the other structured DataTypes supported by the ControlEdge 900 controller OPC UA Server.

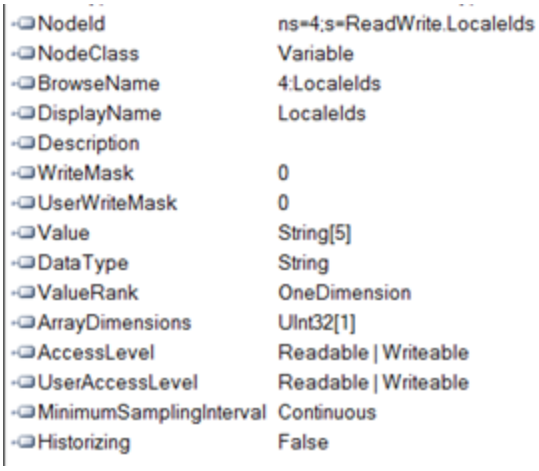
Arrays

The ControlEdge 900 controller array data type is mapped to an OPC UA data type derived from the corresponding elementary data type. The 'ValueRank' attribute is used in UA to provide the information if a value is an array and the 'ArrayDimensions' attribute provides the length of each dimension. Arrays appear as a single node in the UA address space.

Example of an array data type in ControlEdge 900 controller:

```
TYPE
          UaLocaleIds : ARRAY [1..5] OF STRING;
END_TYPE
```

A variable, say 'LocaleIds', of this type in a 900 controller program will be mapped to the OPC data type String with the 'ValueRank' attribute set to 1 and ArrayDimensions[0] set to 5.



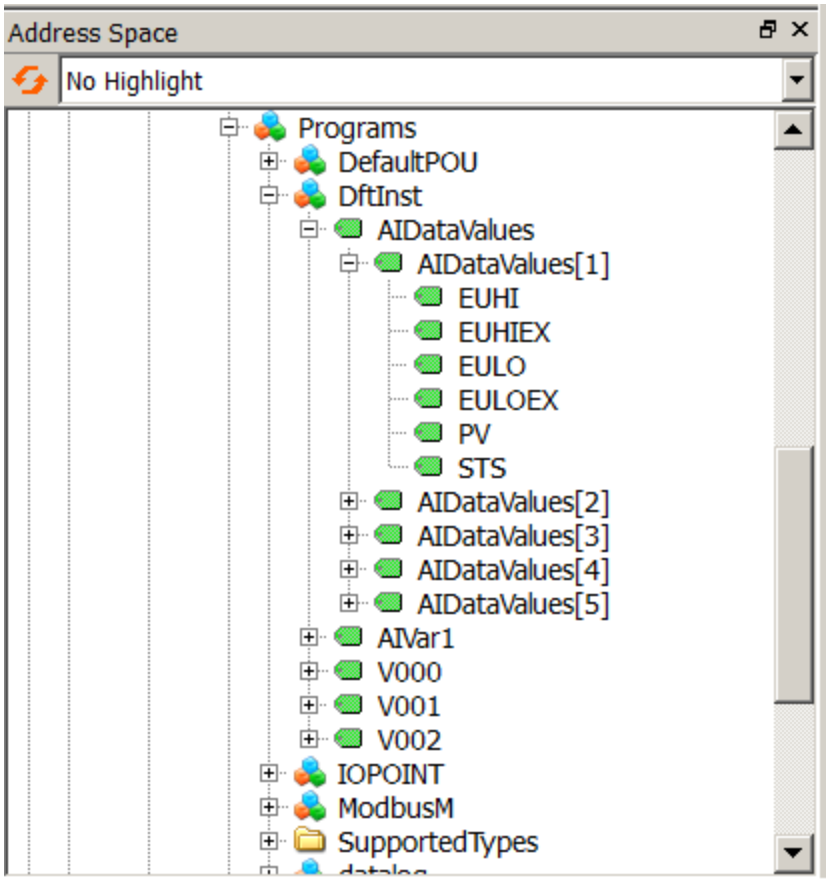
Arrays of Structured types

The ControlEdge 900 controller structured data arrays are modeled along the lines of standard OPC UA array types such as SubscriptionDiagnosticsArray defined in [UA-5]. Unlike an elementary array that appears as a single node in the AddressSpace, the structured data array will expose each entry of the array as a separate node in the AddressSpace. This way a UA Client can access the entire array, read an individual array entry or read individual elements of an array entry.

Shown below is an IEC 61131-3 array of struct ANALOG_INPUT_ TYPE defined in ControlEdge 900 controller.

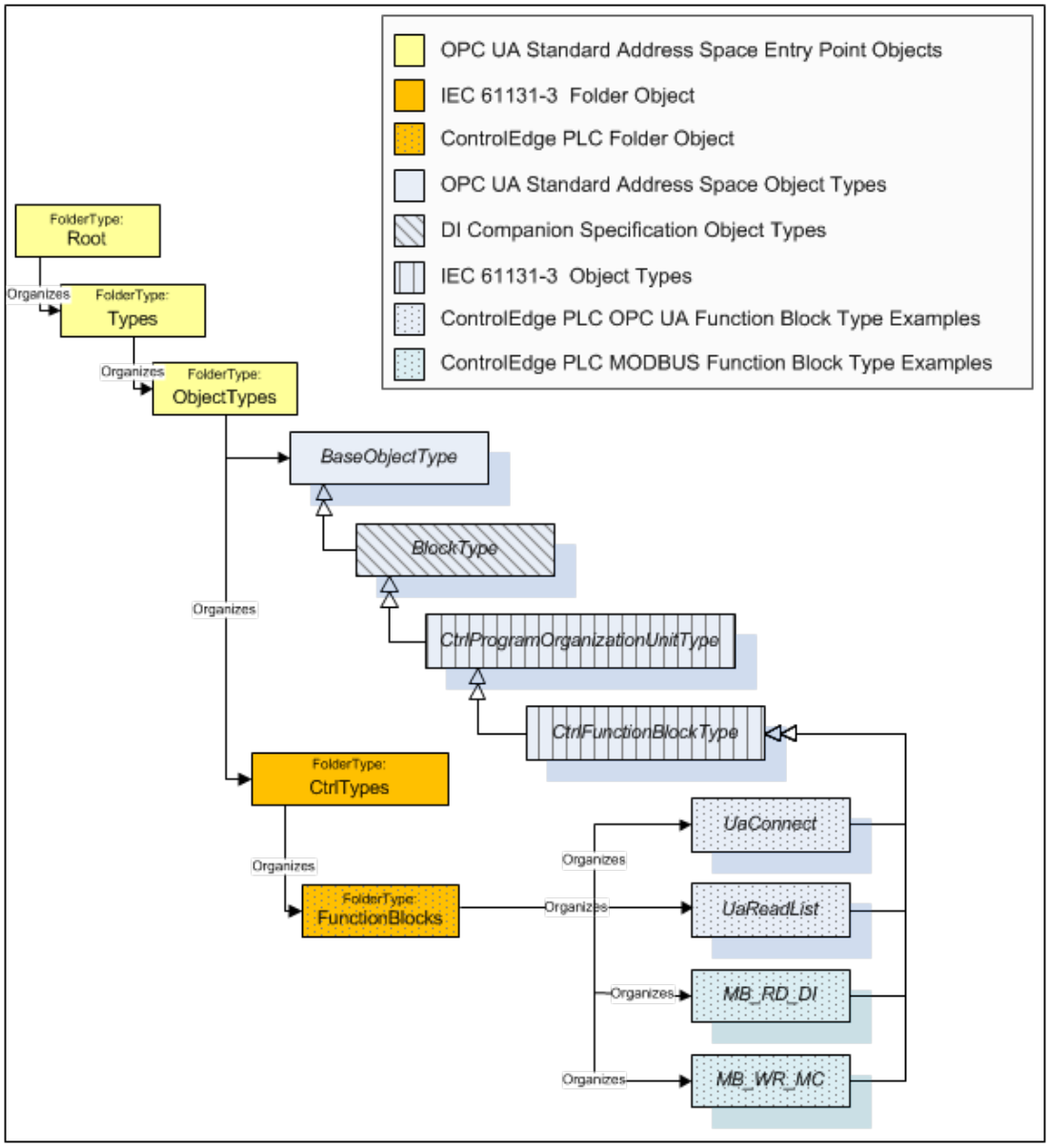
```
TYPE
          AIList : ARRAY [1..5] OF ANALOG_INPUT_TYPE;
END_TYPE
```

The program DftInst has a variable AIDataValues of type AIList. The AddressSpace with the variable AIDataValues is shown below:



Object Types

OPC UA Clients can browse for Server specific function block types from the CtrlTypes\FunctionBlocks Folder object as shown here:



Reference types

The ControlEdge 900 controller OPC UA Server makes use of the following Reference types defined in [PLC]:

- **HasInputVar**– used to reference variables declared with the key word VAR_INPUT
- **HasOutputVar**– used to reference variables declared with the key word VAR_OUTPUT
- **HasInOutVar**– used to reference variables declared with the key word VAR_IN_OUT

- **HasLocalVar**– used to reference variables declared with the key word VAR
- **With**– used to reference the Ctrl Task that executes a Ctrl Program.

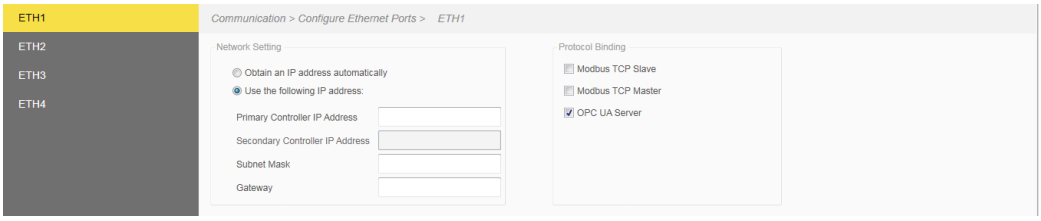
Configure ControlEdge 900 controller OPC UA Server

Configuration

Binding Protocol to Ethernet Ports

You must establish the physical address or endpoint that enables OPC UA client access to the ControlEdge 900 controller OPC UA Server. A maximum of two endpoints can be defined by binding the ETH1 or ETH2 ports on ControlEdge 900 controller to OPC UA Server. One or two endpoints are possible depending on if both ETH1 and ETH2 are bound to OPC UA Server.

1. From the Home Page of ControlEdge Builder, click the arrow beside **Configure Ethernet Ports**, and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the IP address of the Ethernet port.
3. Under **Protocol Binding**, select **OPC UA Server**.



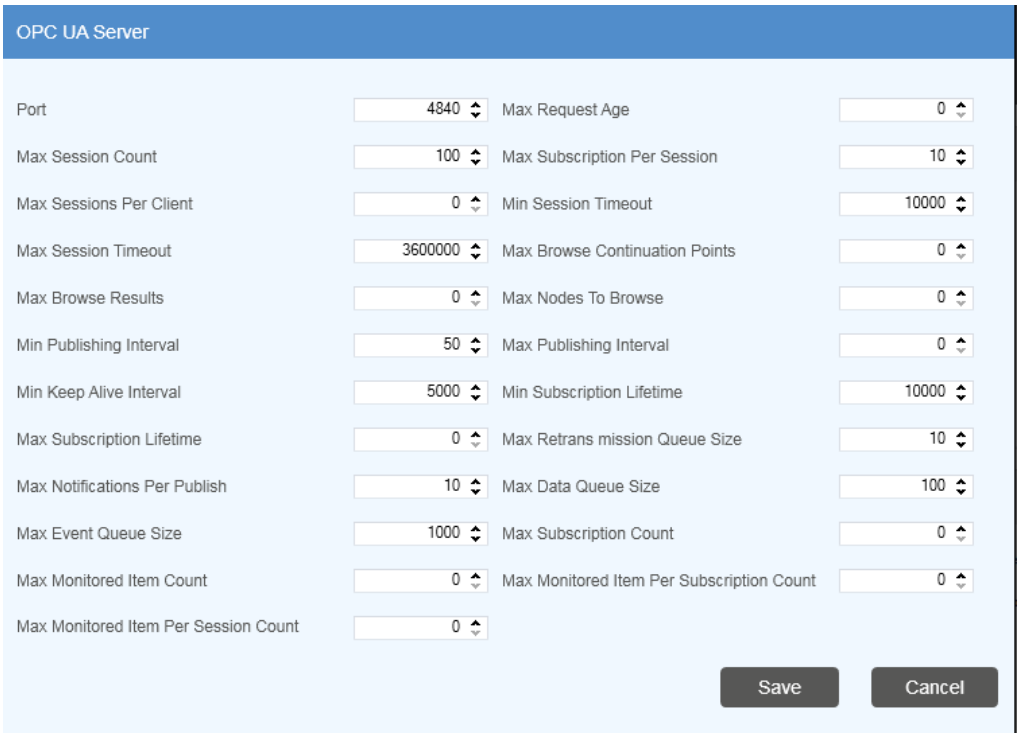
4. Click **Save** to complete the configuration.

Configuring Parameters for OPC UA Server

The ControlEdge 900 controller OPC UA Server supports the UA TCP transport protocol which defaults to communicate on TCP port 4840. This communication port as well as other connectivity and tuning parameters are available for optional configuration. The ControlEdge 900 controller OPC UA server uses default values if no alternative value has been configured. The default configuration is sufficient for getting started with connectivity.

ATTENTION: Make sure that the OPC UA client's time is synchronized to the controller's time.

1. From the Home Page of ControlEdge Builder, click **Configure Protocols** > **OPC UA Server**. The **OPC UA Server** page appears.



2. Configure the parameters if required. See the following table for the parameter description.

   It is recommended to use the default values for the parameters.

   Table 9-1: OPC UA Server parameter description

Parameter	Description
Port	The port that clients will use to connect. For example: opc.tcp://192.168.0.15:**4840** The default value is 4840.
Max Request Age	The maximum age of a request (in milliseconds) the server allows. Zero value is defined as unlimited. The default value, which is 0, indicating that the request age allowed is unlimited.

Parameter	Description
Max Session Count	The maximum number of concurrent sessions the server allows. If you enter a value of 0, the number of sessions allowed is unlimited. The default value is 100.
Max Subscription Per Session	The maximum number of subscriptions allowed by the server for one session. If you enter a value of 0, the number of subscription allowed is unlimited.
Max Session Per Client	The maximum number of concurrent sessions the server allows per client. The default value, which is 0, indicating that the number of sessions allowed is unlimited.
Min Session Timeout	The minimum timeout for a session (in milliseconds). If you enter a value of 0, the minimum timeout is unlimited. The default value is 10000.
Max Session Timeout	The maximum timeout for a session (in milliseconds). If you enter a value of 0, the maximum timeout is unlimited. The default value is 3600000.
Max Browse Continuation Points	The maximum number of browse continuation points managed by a session. The default value is 0.
Max Browse Results	The maximum number of browse results for one browse operation. The default value, which is 0, indicating that the number of browse results is unlimited.
Max Nodes To	The maximum number of nodes to browse.

Parameter	Description
Browse	The default value, which is 0, indicating that the number of nodes allowed is unlimited.
Min Publishing Interval	The minimum cycle rate of the Subscription. The default value is 50 milliseconds.
Max Publishing Interval	The maximum cycle rate of the Subscription. The default value, which is 0, indicating that the publishing interval allowed is unlimited.
Min Keep Alive Interval	The minimum interval after which the subscription sends a notification to the client. This notification ensures the subscription is maintained. The default value is 5000 milliseconds.
Min Subscription Lifetime	Provides assurance to the client that the server is still alive. The minimum period after which the subscription will be deleted if no publish request is received. If you enter a value of 0, the subscription lifetime allowed is unlimited. The default value is 10000 milliseconds.
Max Subscription Lifetime	Provides assurance to the server that the client is still alive. The maximum period after which the subscription will be deleted if no publish request is received. The default value, which is 0 milliseconds, indicating that the subscription lifetime allowed is unlimited.
Max Retransmission Queue Size	The maximum number of messages allowed per Subscription in the republish queue. The default value is 10.
Max	The maximum number of notifications allowed per Publish.

Parameter	Description
Notifications Per Publish	The default value, which is 0, indicating that the number of notifications allowed is unlimited.
Max Data Queue Size	The maximum size of data monitored item queues. The default value is 100.
Max Event Queue Size	The maximum size of event monitored item queues. The default value is 1000.
Max Subscription Count	The maximum number of subscriptions that can be created. The default value, which is 0, indicating that the number of subscriptions allowed is unlimited.
Max Monitored Item Count	The maximum number of items that can be monitored. The default value, which is 0, indicating that the number of items allowed is unlimited.
Max Monitored Item Per Subscription Count	The maximum number of items that can be monitored for each subscription. The default value, which is 0, indicating that the number of items allowed is unlimited.
Max Monitored Item Per Session Count	The maximum number of items that can be monitored for each session. The default value, which is 0, indicating that the number of items allowed is unlimited.
For more information about the parameter descriptions, see the specification in the https://opcfoundation.org/.	

3. Click **Save** to complete the configuration.

Key Parameters to establish OPC UA communication

To establish the communication between OPC UA Sever and OPC UA client, below key parameters of Server must be provided and be required in the configuration in OPC UA side.

Server Endpoint URL

The URL of ControlEdge 900 controller OPC UA Server defined as follows:

<ControlEdge PLC OPC Server URL>:= “opc.tcp://” <IP>”:”<Port>

“opc.tcp://” is the protocol string portion of the URL. This string is constant since the protocol used by the ControlEdge 900 controller OPC UA Server is TCP.

<IP> is the IP address of ETH1 or ETH2 on ControlEdge 900 controller.

<Port> is the port number for the transport protocol. Port number 4840 is the default for OPC UA.

In the following URL examples, the IP address of ETH1 port on ControlEdge 900 controller is set to 192.168.1.10. The IP address of ETH2 port on ControlEdge 900 controller is set to 192.168.2.10.

> **TIP:** One or both URLs may exist depending on the port configuration.

opc.tcp://192.168.1.10:4840

opc.tcp://192.168.2.10:4840

When both Ethernet ports are configured as shown in the example above, the ControlEdge 900 controller OPC UA Server considers the links to be redundant. In this case, the ControlEdge 900 controller OPC Server is listening on both endpoints. When one link is lost, clients can use the URL of the second link to connect to the Server. It is worth noting that the ControlEdge 900 controller OPC UA Server maintains the session created on the failed link until the session timeout period expires after which the session will be deleted.

In the case of redundant ControlEdge 900 controller, the IP address follows the primary controller. Therefore, if a switchover occurs, the client reconnects to the ControlEdge 900 controller OPC UA Server on the new primary with the same URL that was used to connect to the server on the failed primary.

Namespace

OPC UA uses namespaces to uniquely differentiate between the names and IDs it defines and those defined by companion specifications or the local server. The ObjectTypes defined in the UA specification for IEC 61131-3 derive from the OPC UA Device Integration Types which in turn derive from the OPC UA Core ObjectTypes. Thus the ControlEdge 900 controller OPC UA Server includes these 3 namespaces in addition to its own namespace. The list of namespaces used in the Server is shown below:

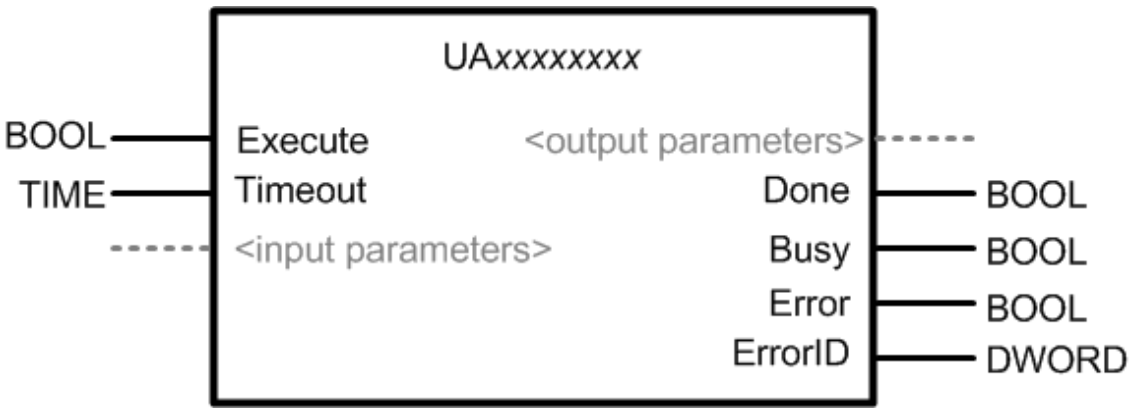
Namespace Index	Namespace	Description
0	http://opcfoundation.org/UA/	Namespace for NodeIds and BrowseNames defined in the OPC UA specification.
1	URL:<IP Address>: Honeywell:ControlEdgePLC:UAServer where IP Address is the IP of the Ethernet port that is bound to OPC UA Server. If UA is enabled on both ETH1 and ETH2, then the IP of ETH1 is used for IP Address.	Namespace index 1 is reserved for the local server, for nodes specific to the server like those shown in section 4.1. Note that this URI is also the ServerURI (appears in index 0 of the ServerArray property). It is also the ApplicationURI in the subjectAltName field of the server's certificate.
2	http://opcfoundation.org/UA/DI/	Namespace for NodeIds and BrowseNames defined in [DI].
3	http://PLCopen.org/OpcUa/IEC61131-3/	Namespace for NodeIds and BrowseNames defined in [PLC].
5	URN: Honeywell:UA:ControlEdgePLC	Namespace for NodeIds and BrowseNames of nodes used to access the underlying ControlEdge PLC data. The exception is when these nodes provide a standard

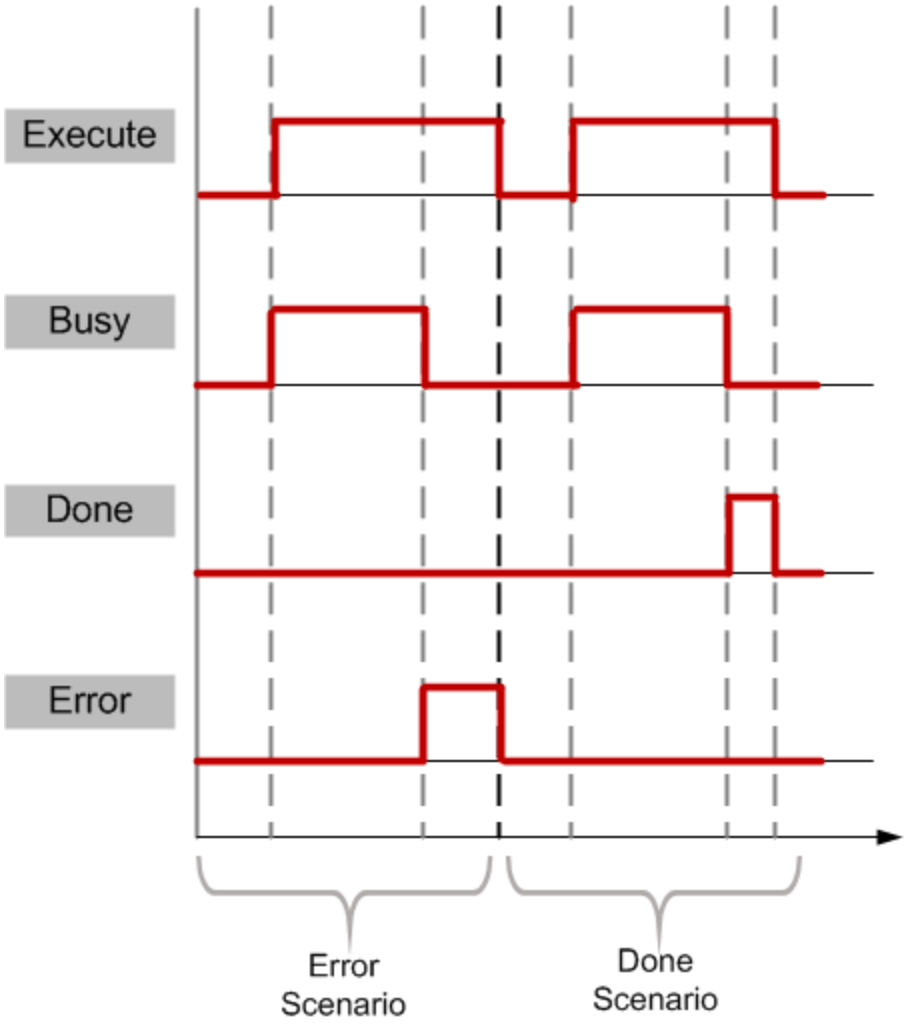
Namespace Index	Namespace	Description
		Property in which case the BrowseName shall have the namespace of the standards body, even though the NodeId will use this namespace. For example, the ParameterSet and the GlobalVars object components of eclrRes shown in section 6.1 - the BrowseName for ParameterSet will use [DI] namespace and the BrowseName for GlobalVars will use the [PLC] namespace. Namespace Uri is used for OPC UA client to get the NameSpaceIndex.

OPC UA Client

IEC 61131-3 OPC UA Function Blocks

The function blocks within the OPCUA library are the native IEC 61131-3 OPC UA Function blocks as defined in [PLC-C]. The figures below illustrate the common interface of each function block regardless of the task that it performs.





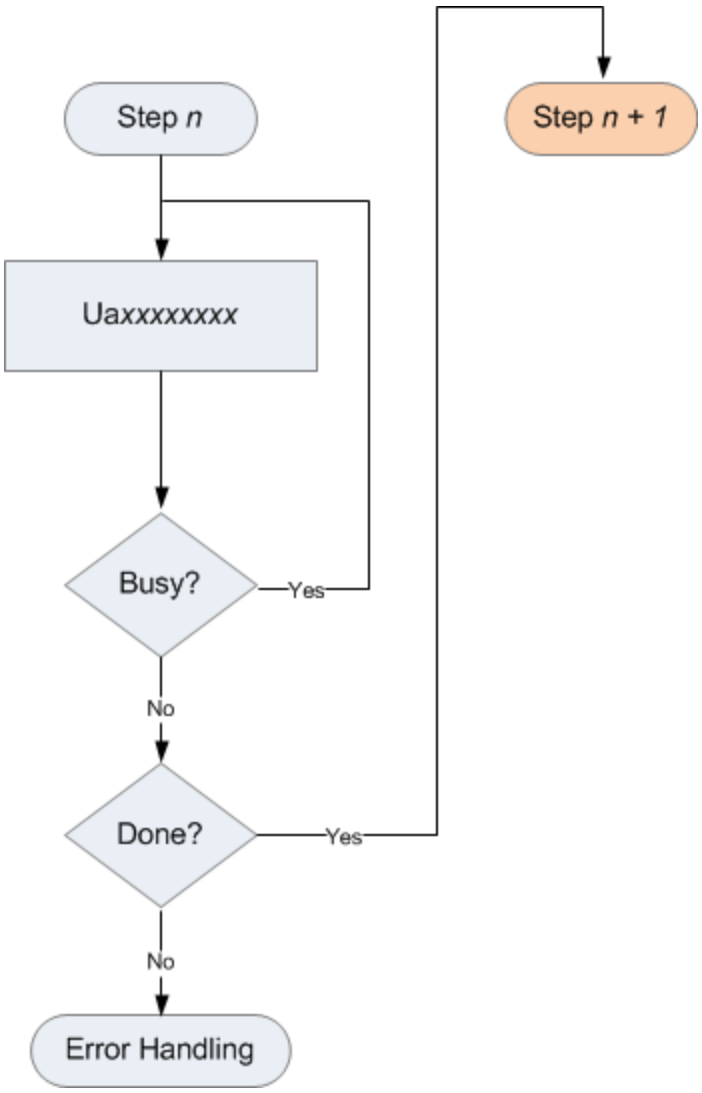
The table below references the above figures to describe the behavior of the IEC 61131-3 OPC UA Function Blocks.

Interface	Behavior
Execute Input	The Function Block command or task is initiated by a rising edge at the **Execute** input. While the value of **Execute** is equal to TRUE, the **Busy**, **Done** and **Error** outputs can be examined independently to determine the status of the function block execution. Furthermore, the **Busy**, **Done** and **Error** outputs are mutually exclusive, i.e. only one of these outputs can be set at a given time.
<input parameters>	Input parameters are read at the rising edge of **Execute** input. Inputs are only read once. Therefore, in order for changes to input parameters to take effect, the **Execute** input must be re-initiated.

Interface	Behavior
Busy Output	The **Busy** output is an indication that the function block has not completed. This output is set to TRUE at the rising edge of **Execute**. It is reset when either **Done** or **Error** is set.
Done Output	The **Done** output, when set, is the indication that the function block has completed successfully. This output is used to trigger the next step in a sequence of function blocks. **TIP:** Once the Done output is TRUE, the Execute input must be reset prior to re-trigger of UaConnect function block.
Error Output	If an error occurs during the execution of the function block, this output is set. The **ErrorID** output contains the error number. Refer to [] for the list of error codes. • Since the function block did not complete successfully, the **Done** output remains reset. • **Timeout** input indicates the maximum time for the Function Block to complete. If the timeout value expires, then the Error output is set.
<output parameters	Output parameters may be invalid while **Busy** output is set. Monitor the **Done** output to trigger valid usage of output parameters (see flowchart diagram below)

IEC 61131-3 OPC UA function block usage

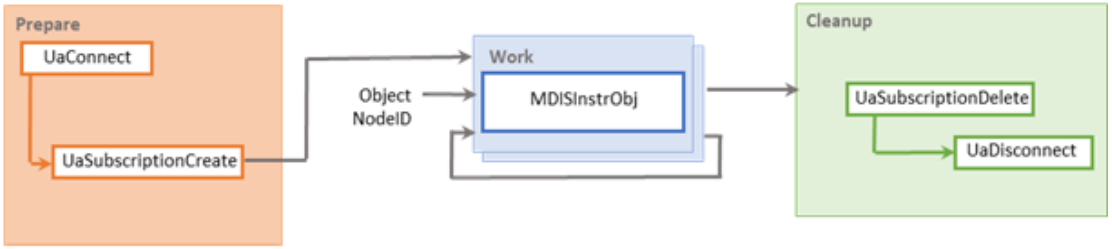
Use of IEC 61131-3 OPC UA function blocks in the OPCUA library requires special handling of BUSY, DONE and ERROR output parameters as shown in the diagram to the right. Failure to perform the special handling of the corresponding output parameters will result in unexpected program errors. Consider use of the OPC UA Helper Function Blocks to facilitate implementation of the required special handling. See next section for details.



MDIS function block library

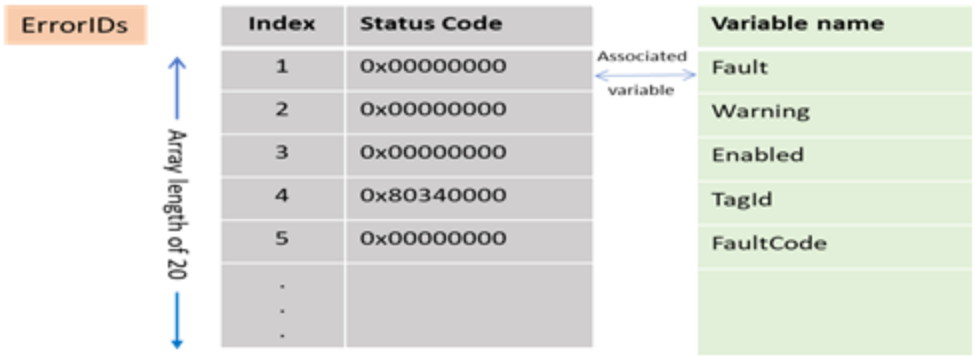
The MDIS library has a set of custom OPC UA function blocks representing all the MDIS OPC UA object types as defined in the MDIS OPC UA Companion Specification V1.2. The MDIS OPC UA Object function blocks are used to obtain data from MDIS OPC UA compliant Servers. For each MDIS object type, the specification identifies a set of data variables as well as method definitions. The MDIS function block library incorporates the data variables into each block as function block parameters or ‘pins’. Separate method function blocks are provided for each of the methods defined in the specification.

When the block's Execute flag is first set and the rising edge is detected, all the data variables of the object get added as Monitored items to a subscription. A subscription handle (obtained from UaSubscriptionCreate) is a required input for all MDIS Object function blocks. The MDIS Object function block must execute (implementation similar to UAMonitoredItem block), in order to retrieve the current value for the data variables of the object. At each rising edge, if a data change notification is available for the object (i.e., one or more variable values have changed), the values are copied to the output pins representing the data variables and the Done flag is set. The Busy flag is set while the block waits for a data change notification to become available. The Error flag is set if any problem was encountered and the ErrorID output pin will hold the associated error code. Note that Busy and Error/Done are mutually exclusive. That is, if Busy is set then Error/Done will not be set. Conversely, if either Done or Error is set, then Busy will not be set. The following block diagram shows the sequence of operations required to get data updates for a MDIS object.



Every MDIS Object Function block will have an ErrorIDs pin which is an array of DWORD. Each element of this array represents the status of a data variable of the object. It provides status on whether a data variable was successfully added to the subscription. Since many of the data variables defined in the MDIS specification are optional, not all servers will support all the variables for an object. The MDIS function blocks will attempt to add all the defined variables, including optional ones, to the subscription. If the server does not implement the optional variable, then a BadNodeIdUnknown (0x80340000) error status will be shown at the array index associated with the variable. For those variables that were successfully added, on receiving data change notifications, the ErrorIDs are updated to report the quality of the data value of the variable. Status code values which are not 0x0 indicate that corresponding data variable's value is not useable. The list of possible status codes can be found in the 'OPC UA Error code Reference' section.

The internal representation of the ErrorIDs parameter is shown below. The variable associated with each index for an object FB is fixed and is listed in the parameter section of each function block.



The MDIS Object function blocks are:

- MDISDiscrtInstrObj
- MDISDigInstrObj
- MDISInstrObj
- MDISChokeObj
- MDISValveObj

In addition to the function blocks that represent the objects themselves, there are function blocks for every method that can be called on these objects. All method FBs require a connection handle and Object NodeID as inputs.

The method function blocks are listed below:

- MDISObjEnableDisable
- MDISDiscrtInstrWriteVal
- MDISDigInstrWriteState
- MDISInstrWriteValue
- MDISChokeMove
- MDISChokeStep

- MDISChokeAbort
- MDISChokeSetCalcPos
- MDISValveMove

For more information on MDIS function block, see "MDIS function block" in *ControlEdge Builder Function and Function Block Configuration Reference Guide*.

Usage Considerations

The following table outlines a typical usage scenario when combining OPC UA function blocks. Use the table to map the program phase tasks to function blocks from the OPCUA or Honeywell Helper libraries. See Prepare Phase for Reads, Writes or Method Calls for more information.

Program Phase	Task	OPC UA Library	Honeywell Helper Library
Prepare	Establish connection to UA Server	UaConnect	HonUaConnectSecurityNone
	Get the namespace-index of a namespace-URI for the variables in the target OPC UA server's address space to be read or written	UaNamespaceGetIndex (translatepath)	See note**1**
	Get the node handle for the variables in the target OPC UA servers address space to be read or	UaNodeGetHandle UaNodeGetHandleList	See note**2**

Program Phase	Task	OPC UA Library	Honeywell Helper Library
	written		
	Get the method handle for a method call	UaMethodGetHandle	See note**3**
	Translate Path - Get node parameters using path of the node.	UaTranslatePath UaTranslatePathList UaTranslatePaths	HonUaTranslatePathList
	Create a subscription	UaSubscriptionCreate	See note **4**
Work	Reading of variables in the namespace of UA Server	UaRead UaReadList	HonUaReadNode HonUaReadNodeList
	Writing of variables in the namespace of UA server	UaWrite UaWriteList	HonUaWriteNode HonUaWriteNodeList
	Execution of methods supported by UA Server	UaMethodCall	HonUaCallMethod
	Use a subscription to monitor variables	UaMonitoredItemAdd UaSubscriptionOperate	HonUaSubscribeNode
Cleanup	Release node handle	UaNodeReleaseHandle UaNodeReleaseHandleList	See note**5**
	Release the method	UaMethodReleaseHandle	

Program Phase	Task	OPC UA Library	Honeywell Helper Library
	handle		
	Remove monitored variables from a subscription	UaMonitoredItemRemove	See note**6**
	Release a subscription	UaSubscriptionDelete	
	Terminate the connection to OPC UA Server	UaDisconnect	HonUaConnectSecurityNone
Utilities	Monitor Handle to signal loss of handle due to ControlEdge PLC reset.	See note**7**	HonUaHandleDetector
	Utility block to monitor and signal a change in state.	See note**8**	HonUaStateDetector
	Converts a variable with variant data type to string format. This is useful for debugging purposes.		HonUaVariantToString

Note:

1. Currently, there is no Honeywell Helper Function Block for UaNamespaceGetIndex. However, refer to the structured text code for HonUaTranslatePathList for an example.

Program Phase	Task	OPC UA Library	Honeywell Helper Library
2.	Currently, there are no stand-alone Honeywell Helper Function Blocks for UaNodeGetHandle and UaNodeGetHandleList. HonUaReadNode and HonUaWriteNode are designed to include UaNodeGetHandle. HonUaReadNodeList and HonUaWriteNodeList are designed to include UaNodeGetHandleList.		
3.	Currently, there is no stand-alone Honeywell Helper Function Block for UaMethodGetHandle. This is included in HonUaCallMethod.		
4.	Currently there is no stand-alone Honeywell Helper Function Block for simply creating a subscription.		
5.	Currently, there are no Honeywell Helper Function Blocks for UaNodeReleaseHandle, UaNodeReleaseHandleList and UaMethodReleaseHandle.		
6.	Currently, there are no Honeywell Helper Function Blocks for UaMonitoredItemRemove and UaSubscriptionDelete.		
7.	There are no IEC 61131-3 OPC UA blocks defined for these funtions.		
8.	There are no IEC 61131-3 OPC UA blocks defined for these funtions.		

There are several ways to facilitate development of a PLC program with OPC UA function blocks:

- Understand the OPC UA Function Block state model (Execute-Busy-Done).
- Connect a graphical OPC UA Client to the target OPC UA server (see tip below). Browse the target server address space to become familiar with OPC UA communication parameters including:
  - Node Identifier
  - Node Namespace Index and associated Namespace URI
  - Node data type
  - Method Node ID and Object Node ID
- Use the Honeywell Helper Function Blocks. These function blocks are implemented with structured text. Each helper function block can be used directly in the same way function blocks from OPCUA Library are used. Alternatively, use the Honeywell Helper Function Blocks as examples to create custom helper function blocks.

TIP: Use a graphical OPC UA Client

Consider use of a graphical OPC UA Client to access attributes of the target OPC UA server and the nodes within its address space. The example below shows the Browse output from OPC Foundation Sample Client for an individual node.

NodeId	ns=4;s=ReadWrite.LocaleIds
NodeClass	Variable
BrowseName	4:LocaleIds
DisplayName	LocaleIds
Description	
WriteMask	0
UserWriteMask	0
Value	String[5]
DataType	String
ValueRank	OneDimension
ArrayDimensions	UInt32[1]
AccessLevel	Readable | Writeable
UserAccessLevel	Readable | Writeable
MinimumSamplingInterval	Continuous
Historizing	False

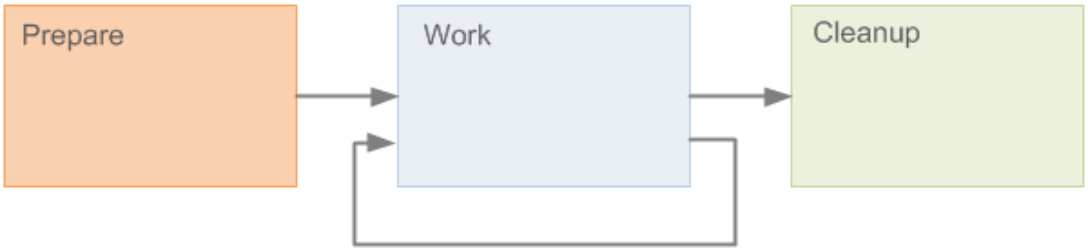
To determine the NodeId attributes (NamespaceIndex, IdentifierType and Identifier), view the value to the right of the NodeId object in the Browse output. In this example:

NamespaceIndex = 4

IdentifierType = String

Identifier = ReadWrite.LocaleIds

The following sections describe the detailed usage information to perform work tasks such as UaRead, UaReadList, UaWrite, UaWriteList or UaMethodCall. These sections follow the order of tasks introduced in the table above (shown below graphically).



Prepare Phase for Reads, Writes or Method Calls

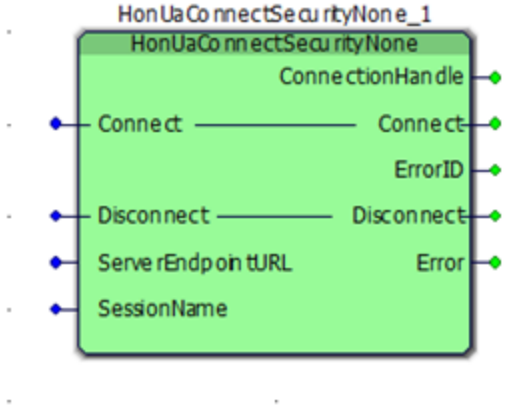
Establishing a connection to the target OPC UA server is the primary objective of the Prepare Phase. OPC UA servers require a physical network location or endpoint URL for connectivity. The endpoint URL for an OPC UA server consists of several parts: prefix, host and port. “opc.tcp://” is an example of a valid prefix portion of the URL. It identifies the endpoint as an OPC Server and the protocol. The host portion is either a hostname or IP address. The port number is the target port of the OPC UA Server, which may vary among OPC UA Server providers. Consult with the target OPC UA Server documentation to verify the endpoint URL.

A new session is established each time a connection is established.

Establishing Connection with HonUaConnectSecurityNone

This function block uses the UaConnect function block to establish an OPC UA session to a remote OPC Server using a specified Server URL and Session name. The security related fields of SessionConnectInfo are set to values that indicate no use of security. If successfully established, the named session will have a 30 seconds timeout.

Figure 9-2: HonUaConnectSecurityNone



VAR_INPUT		
ServerEndpointURL	STRING	e.g., “opc.tcp://192.168.1.30:51210/UA/SampleServer”
SessionName	STRING	Each time Connect executes a new session is created on the server. This name will be associated with that session

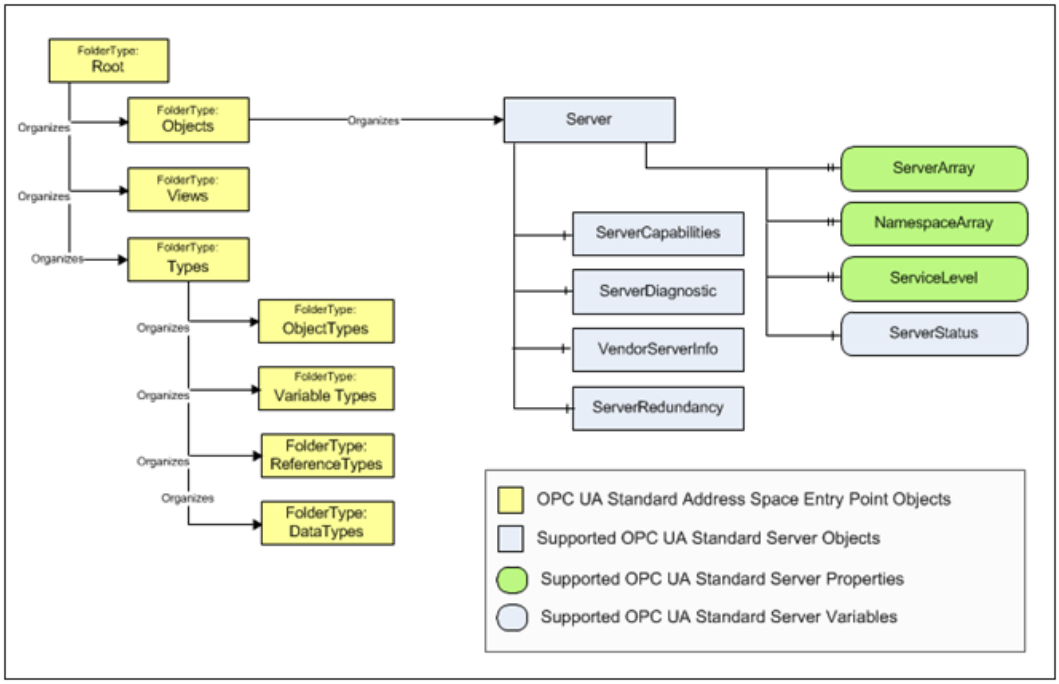
VAR_OUTPUT		
ConnectionHandle	DWORD	The handle associated with this connection. Handle is valid until Disconnect is set.
Error	BOOL	If set, signals an error occurred when attempting to connect
ErrorID	DWORD	Error ID if any, returned by the server

VAR_IN_OUT		
Connect	BOOL	When set TRUE and if ConnectionHandle is zero, initiates a new connection. Upon completion of 1 connection attempt (successful or unsuccessful) will automatically reset to FALSE.
Disconnect	BOOL	Set to FALSE

Use the HonUaconnectSecurityNone function block as an example to create a custom connect helper block if alternative values for non-security related fields of SessionConnectInfo are required (such as LocalIDs or SessionTimeout).

Accessing the Address Space of target OPC UA Server

The information that the target OPC UA Server makes available to clients is referred to as its address space. The elements of the address space are represented as a set of nodes. Refer to [OPC-3] to address space concepts including nodes, node attributes and interconnections. Using standard OPC UA notation, the diagram below shows the set of nodes common to all OPC UA Servers. As indicated by the diagram key, this set of nodes includes objects, variables, and properties. The diagram also illustrates the relationships between nodes. Use a graphical OPC UA client connected to the target OPC UA Server to view the set of standard nodes for the target OPC UA Server.

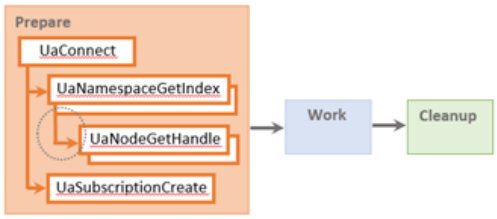


Every OPC UA node, regardless of its node type (e.g. object, variable, property, etc.) is represented by a node identifier, which consists of the following:

Element	NodeId
NamespaceIndex	1
IdentifierType	String
Identifier	Instrument_01.Temp

OPC UA uses namespaces to uniquely differentiate between the names and IDs it defines and those defined by OPC UA companion specifications (e.g. FDI) or the target OPC UA server itself. In the example NodeId shown above, the NamespaceIndex is 1 which is the index reserved for the “local” server. NamespaceIndex 0 is reserved for OPC Foundation. It is the index to NodeIds and BrowseNames defined in the OPC UA specifications. Consult with the target OPC UA Server documentation for a list of supported name spaces. Alternatively, use a graphical OPC UA Client connected to your target OPC Server, to browse to the NameSpaceArray property node (see diagram above). This node contains the required information for the registered Namespaces on the target OPC UA Server. Each index in

the array is associated with a Namespace URI. Use the Namespace URI with **UaGetNamespaceIndex** function block to resolve each Namespace index from your PLC program. Since a namespace index can change dynamically, best practice is to resolve the namespace URI programmatically.

Prepare with base UA blocks	Prepare with Honeywell Helper UA blocks
	
The input to UaNodeGetHandle (as circled above) requires steps external to PLC program to obtain the nodeIDds within each registered namespace of the target OPC UA Server. • Consult the target OPC UA documentation • Use graphical OPC UA client to browse the address space • Requires special handling to ensure completion of base UA block before passing input to subsequent block. • **UaNodeGetHandle**maps target OPC Server nodeId to a node handle maintained by ControlEdgePLC OPC UA Client. Node handles must be explicitly supplied to the base read and write blocks.	The input to HonUaTranslatePathList (as circled above) requires steps external to PLC program to obtain starting NodeId and a list of relative paths. This information is used by the target OPC Server to obtain the nodeIds for variables to read or write. HonUaTranslatePathList optionally performs the dynamic resolution to a given namespace index. See Obtaining NodeIds with HonUaTranslatePathList for more information. **TIP: HonUaReadNode**, **HonUaReadNodeList**, **HonUaWriteNode**, **HonUaWriteNodeList** and **HonUaSubscribeNode** use NodeIds rather than node handles as input. The mapping to node handles is built in to these Honeywell helper function blocks.
The input to **UaConnect** and **HonUaConnectSecurityNone** requires steps external to PLC program to obtain the URL of the target OPC UA Server.	

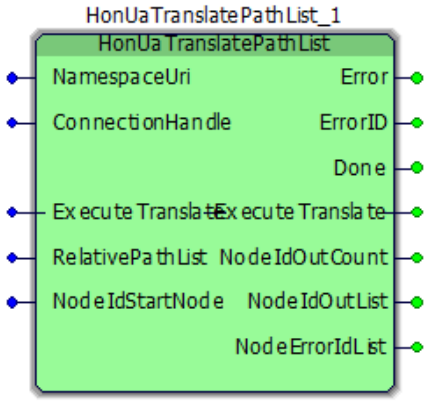
Obtaining NodeIds with HonUaTranslatePathList

HonUaTranslatePathList is a convenient way to get NodeIds within a single namespace registered with the target OPC UA server. It uses **UaNamespaceGetIndex** and **UaTranslatePaths**.

This function block requires the nodeID as a starting point in the address space of the target OPC UA server. **HonUaTranslatePathList** optionally resolves the namespace index in relative paths. If the substitution token '#' is inserted into the relative paths in RelativePathList then the block acquires the index of this Uri from namespace table of the target server. It then substitutes that index at each '#'.

For example, if a string in the RelativePathList is "/#:Drum1001/#:LIX001/#:Output" and NameSpaceUri "http://opcfoundation.org/sampleserver" is located at index 4 in the namespace index table of the target server, then HonUaTranslatePathList modifies the string to "/4:Drum1001/4:LIX001/4:Output" prior to passing to the target server for translation.

Figure 9-3: HonUaTranslatePathList



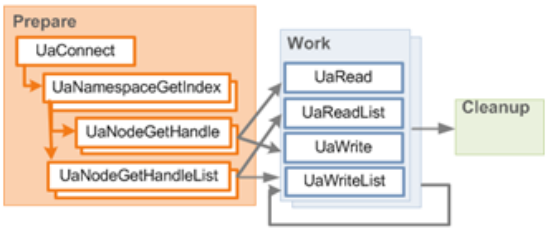
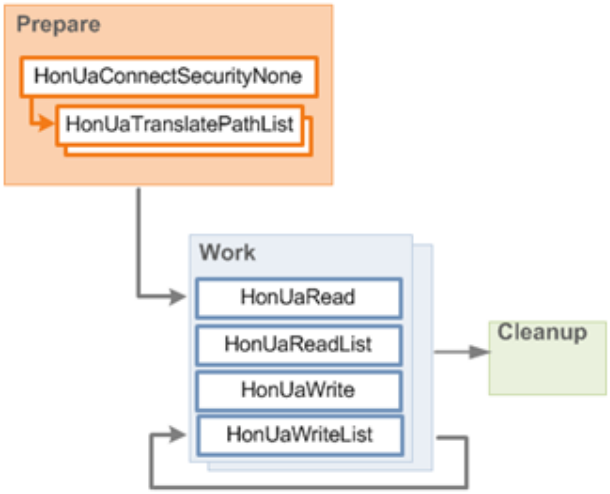
VAR_INPUT		
ConnectionHandle	DWORD	Connection handle obtained from Connection block (e.g., "Connect_SecurityNone" above)
NodeIdStartNode	UaNodeID	The RelativePathList is evaluated using this node as a starting point.
RelativePathList	String255List	Relative paths to the target nodes using NodeIdStartNode as a starting point. See above

VAR_INPUT		
		for syntax.
NamespaceUri	STRING	Supplied if NamespaceIndex substitution is desired in any Relative Path. Otherwise, may be set to empty string.

VAR_OUTPUT		
Error	BOOL	If set, signals an error occurred when attempting to translate paths
ErrorID	DWORD	Error ID if any, returned by the server
Done	BOOL	Flag indicating that the function block execution has completed. This flag will be reset FALSE the next time ExecuteTranslate is set TRUE.
NodeIdOutCount	UINT	Number of NodeIDs returned
NodeIdOutList	UaNodeIDList	Node IDs corresponding to the relative paths in RelativePathList
NodeErrorIdList	UaNodeIDList	Error ID associated with translating the corresponding relative path to a Node ID. Note that ErrorID above will be set if any element of this list has a status other than good.

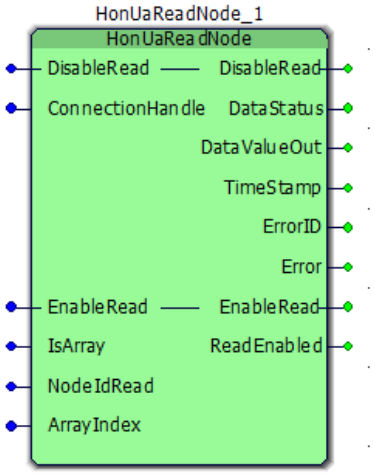
VAR_IN_OUT		
ExecuteTranslate	BOOL	When set TRUE, initiates the relative path to NodeID translation. Upon completion of 1 such attempt (successful or unsuccessful) will automatically reset to FALSE.

Work Phase - Read/Write/Method Call

Work Phase with base UA blocks	Work Phase with Honeywell Helper UA blocks
 **Tip:** Requires special handling to ensure completion of base UA block before passing input to subsequent block	 **TIP:** Node handle mapping is built-in to **HonUaRead**, **HonUaReadList**, **HonUaWrite** and **HonUaWriteList**

Reading a single variable

Figure 9-4: HonUaReadNode



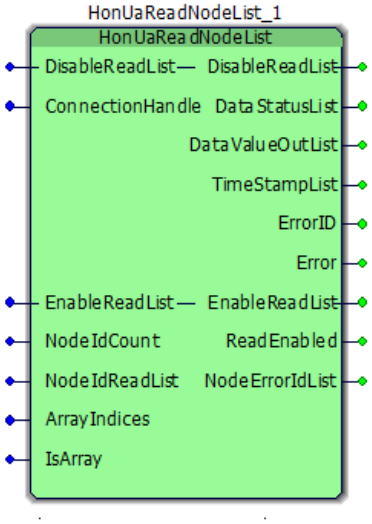
VAR_INPUT		
ConnectionHandle	DWORD	Connection handle obtained from Connection block (e.g., “Connect_SecurityNone” above)
NodeIdRead	UaNodeID	Node ID whose data value is to be read.
IsArray	BOOL	Flag indicating whether or not the NodeIdRead data value is an array.
ArrayIndex	UINT	If IsArray is TRUE then this identifies the array index to read.

VAR_OUTPUT		
DataStatus	UDINT	Status code associated with the DataValueOut
DataValueOut	UAVariant	Value of the node (attribute 13)
TimeStamp	UADateTime	Source timestamp associated with DataValueOut
ErrorID	DWORD	Error ID if any, returned by the server when attempting to invoke the Read service.
Error	BOOL	If set, signals that an error occurred when attempting to invoke the Read service .
ReadEnabled	BOOL	When set, indicates that block is enabled and read service will be called with each task cycle.

VAR_IN_OUT		
EnableRead	BOOL	When set TRUE, enables this read block. Read service will be called with each task cycle. See ReadEnabled above to verify that block is enabled.
DisableRead	BOOL	When set TRUE, disables this read block. Read service will not be called with each task cycle. See ReadEnabled above to verify that block is disabled.

Reading a list of variables

Figure 9-5: HonUaReadNodeList



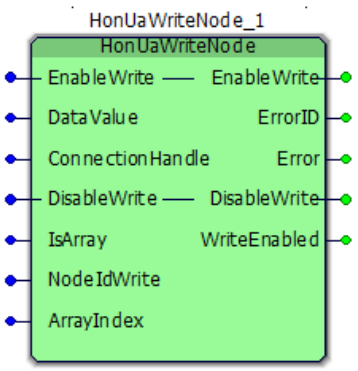
VAR_INPUT		
ConnectionHandle	DWORD	Connection handle obtained from Connection block (e.g., “Connect_SecurityNone” above)
NodeIdCount	UINT	The number of Node IDs in NodeIdReadList
NodeIdReadList	UaNodeIDList	Node identifiers of the nodes whose values are to be read by this block (max 20 identifiers).
IsArray	BOOL	Flag indicating whether or not the NodeIdReadList data values are arrays
ArrayIndices	UINTList	If IsArray is TRUE then this identifies the array index for each data value of NodeIdReadList to read. NodeIdReadList and ArrayIndices must contain the same number of elements.

VAR_OUTPUT		
ErrorID	DWORD	Error ID if any, returned by the server when attempting to invoke the Read service.
Error	BOOL	If set, signals that an error occurred when attempting to invoke the Read service
ReadEnabled	BOOL	When set, indicates that block is enabled and the Read service will be called with each task cycle.
DataStatusList	UaDWORDList	Status code associated with corresponding value of the DataValueOutList
DataValueOutList	UAVariantList	Value of each node (attribute 13)
TimeStampList	UaDateTimeList	Source Timestamp associated with corresponding value of the DataValueOutList
NodeErrorIdList	UaDWORDList	Error ID associated with corresponding value of the DataValueOutList. Note that ErrorID above will be set if any element of this list has a status other than good.

VAR_IN_OUT		
EnableReadList	BOOL	When set TRUE, enables this read block. Read service will be called with each task cycle. See ReadEnabled above to verify that block is enabled.
DisableReadList	BOOL	When set TRUE, disables this read block. Read service will not be called with each task cycle. See ReadEnabled above to verify that block is disabled.

Writing a single variable

Figure 9-6: HonUaWriteNode



VAR_INPUT		
ConnectionHandle	DWORD	Connection handle obtained from Connection block (e.g., “Connect_SecurityNone” above)
NodeIdWrite	UaNodeID	Node ID whose data value is to be written.
IsArray	BOOL	Flag indicating whether or not the NodeIdWrite data value is an array
ArrayIndex	UINT	If IsArray is TRUE then this identifies the array index to write.
DataValue	UAVariant	Value to be written (attribute 13)

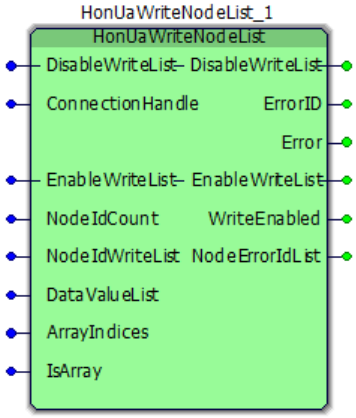
VAR_OUTPUT		
ErrorID	DWORD	Error ID if any, returned by the server when attempting to invoke the Write service.
Error	BOOL	If set, signals that an error occurred when attempting to invoke the Write service
WriteEnabled	BOOL	When set, indicates that block is enabled and write service will be called with each task cycle.

VAR_IN_OUT		
EnableWrite	BOOL	When set TRUE, enables this write block. Write service will be called with each task cycle. See WriteEnabled above to verify that block is enabled.
DisableWrite	BOOL	When set TRUE, disables this write block. Write service will not

VAR_IN_OUT		
		be called with each task cycle. See WriteEnabled above to verify that block is disabled.

Writing a list of variables

Figure 9-7: HonUaWriteNodeList



VAR_INPUT		
ConnectionHandle	DWORD	Connection handle obtained from Connection block (e.g., “Connect_SecurityNone” above)
NodeIdCount	UINT	The number of Node IDs in NodeIdWriteList
NodeIdWriteList	UaNodeIDList	Node identifiers of the nodes whose values are to be written by this block (max 20 identifiers).
IsArray	BOOL	Flag indicating whether or not the NodeIdWriteList data values are arrays
ArrayIndices	UINTList	If IsArray is TRUE then this identifies the array index for each data value of NodeIdWriteList to read. NodeIdWriteList and ArrayIndices must contain the same number of elements.
DataValueList	UAVariantList	Values to be written (attribute 13).

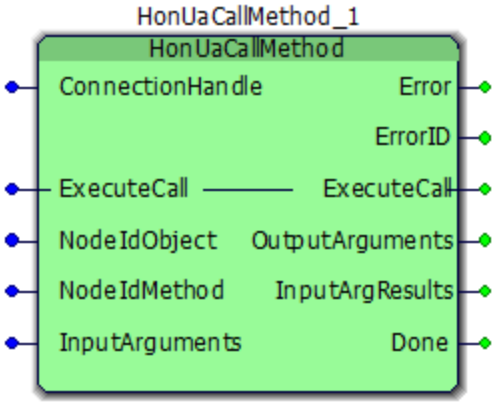
VAR_OUTPUT		
ErrorID	DWORD	Error ID if any, returned by the server when attempting to invoke the Write service.

VAR_OUTPUT		
Error	BOOL	If set, signals that an error occurred when attempting to invoke the Write service
WriteEnabled	BOOL	When set, indicates that block is enabled and the Write service will be called with each task cycle.
NodeErrorIdList	UaDWORDList	Error ID associated with corresponding value of the DataValueList when attempting to write the value. Note that ErrorID above will be set if any element of this list has a status other than good.

VAR_IN_OUT		
EnableWriteList	BOOL	When set TRUE, enables this write block. Write service will be called with each task cycle. See WriteEnabled above to verify that block is enabled.
DisableWriteList	BOOL	When set TRUE, disables this write block. Write service will not be called with each task cycle. See WriteEnabled above to verify that block is disabled.

Calling a Method

Figure 9-8: HonUaCallMethod



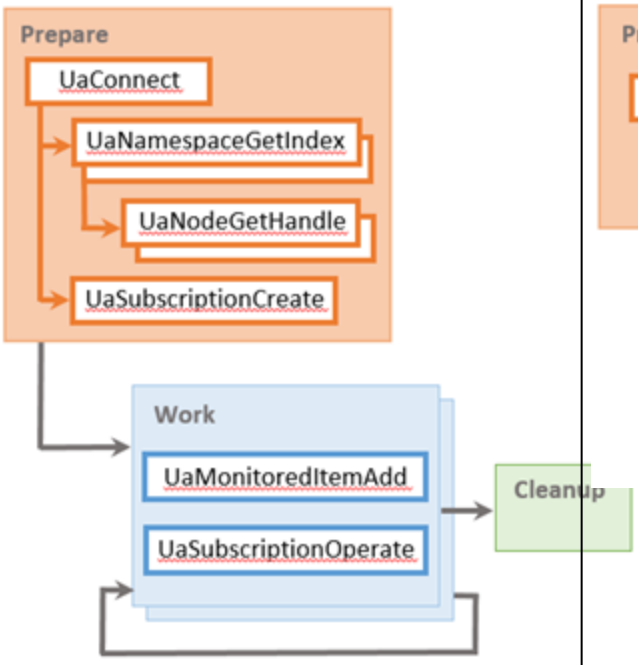
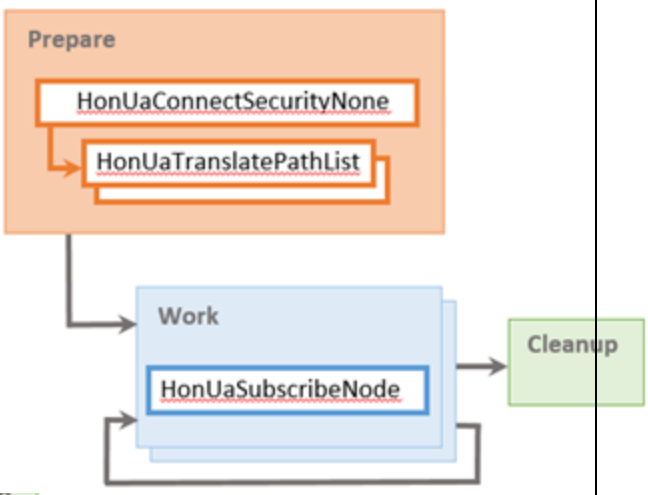
VAR_INPUT		
ConnectionHandle	DWORD	Connection handle obtained from Connection block (e.g., “Connect_ SecurityNone” above)
NodeIdentifierObject	UaNodeID	Node ID of the object node whose method is

VAR_INPUT		
		to be called by this block
NodeIdentifierMethod	UaNodeID	Node ID of the method node to be called by this block
InputArguments	UAVariantList	Input arguments for this method. Note that some methods may not require any input arguments.
Done	BOOL	Flag indicating that the method call has completed. This flag will be reset FALSE the next time ExecuteCall is set TRUE.

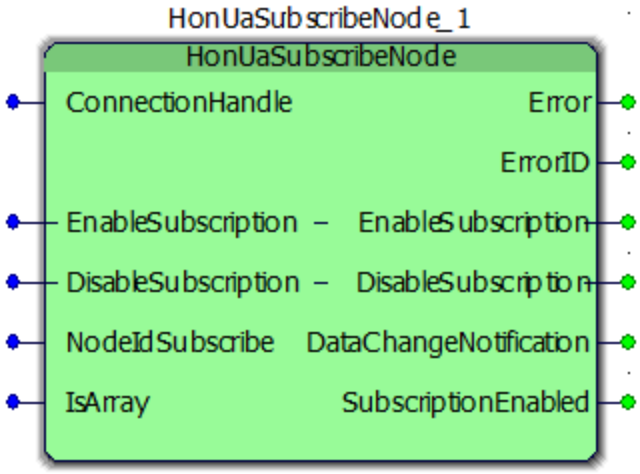
VAR_OUTPUT		
ErrorID	DWORD	Error ID if any, returned by the server when attempting to invoke the Call service.
Error	BOOL	If set, signals that an error occurred when attempting to invoke the Call service
OutputArguments	UAVariantList	Output arguments returned by this method. Note that some methods may not return output arguments
InputArgResults	UaDWORDList	Status code associated with each argument in the InputArguments.

VAR_IN_OUT		
ExecuteCall	BOOL	When set TRUE, invokes the method call. Upon completion of 1 method call attempt (successful or unsuccessful) will automatically reset to FALSE.

Work Phase - Subscripbe for Variable Notifications

Work Phase with base UA blocks	Work Phase with Honeywell Helper UA blocks
	
TIP: Requires special handling to ensure completion of base UA block before passing input to subsequent block.	**TIP:** Node handle mapping is built-in to **HonUaSubscribeNode**.

Subscribing for single variable notifications



VAR_INPUT		
ConnectionHandle	DWORD	Connection handle obtained from Connection block (e.g., “Connect_SecurityNone” above)
NodeIdSubscribe	UaNodeID	The NodeId of the data variable node which will be added as monitored item to the subscription.
IsArray	BOOL	Flag indicating whether or not the NodeIdSubscribe data value is an array.

VAR_OUTPUT		
ErrorID	DWORD	Error ID if any, returned by the server when attempting to invoke the subscription or monitored item service.
Error	BOOL	If set, signals that an error occurred when attempting to invoke the subscription or monitored item service.
SubscriptionEnabled	BOOL	A flag indicating that the subscription is currently enabled.
DataChangeNotification	UaDataChangeNotification	Notifications for the subscribed node. A notification will occur when

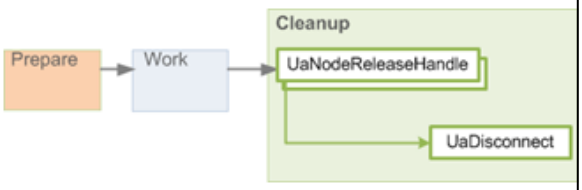
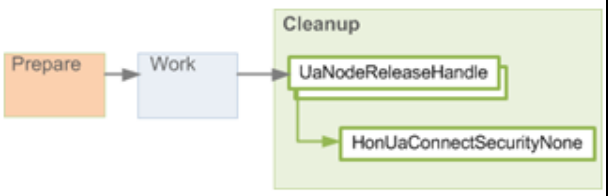
VAR_OUTPUT		
		the value or the status of the variable changes.

VAR_IN_OUT		
EnableSubscription	BOOL	Set the subscription enabled.
DisableSubscription	BOOL	Set the subscription disabled.

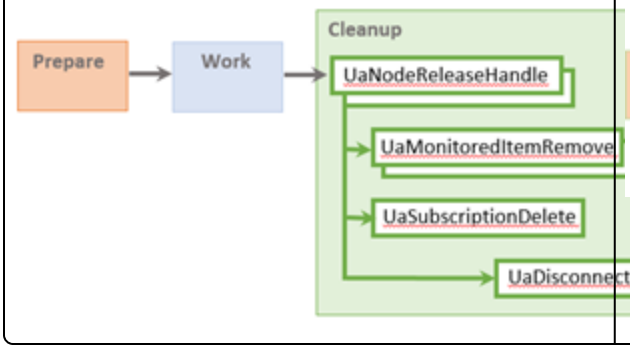
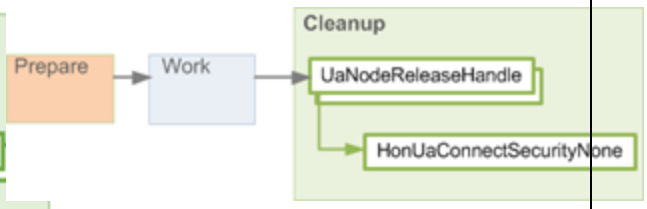
Cleanup Phase

Releasing resources that were previously acquired during the Prepare and Work Phases is the primary objective of the Cleanup Phase.

Cleanup - Read/Write/Method Call;

Cleanup with base UA blocks	Cleanup with Honeywell Helper UA blocks
	

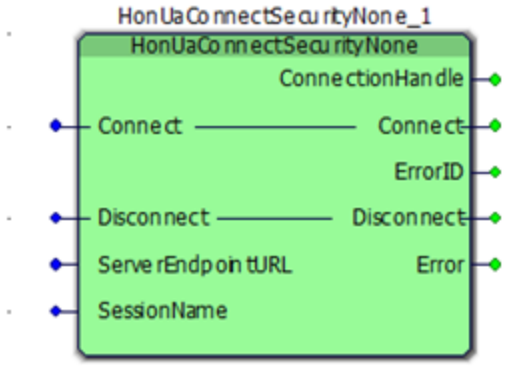
Cleanup - Subscribe for Variable Notifications

Cleanup with base UA blocks	Cleanup with Honeywell Helper UA blocks
	

Terminate Connection with HonUaConnectSecurityNone

This function block uses the UaDisconnect function block to terminate an OPC UA session to a remote OPC Server using the ConnectionHandle.

Figure 9-9: Terminate Connection with HonUaConnectSecurityNone



VAR_INPUT		
ServerEndpointURL	String255	e.g., "opc.tcp://192.168.1.30:51210/UA/SampleServer"
SessionName	STRING	Each time Connect executes a new session is created on the server. This name will be associated with that session

VAR_OUTPUT		
ConnectionHandle	DWORD	The handle associated with this connection. Handle is valid until Disconnect is set.
Error	BOOL	If set, signals an error occurred when attempting to connect
ErrorID	DWORD	Error ID if any, returned by the server

VAR_IN_OUT		
Connect	BOOL	Set to FALSE
Disconnect	BOOL	When set TRUE initiates a disconnect of the current ConnectionHandle (as indicated by ConnectionHandle). Upon completion of 1 disconnect attempt (successful or unsuccessful) will automatically reset to FALSE.

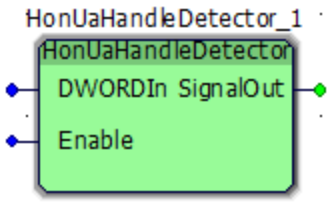
Utilities

In addition to the helper blocks described in the previous Prepare and Work phase sections, the Honeywell OPC UA Helper Block library includes several utilities that are convenient for error detection, error handling and debugging.

Monitoring the target OPC UA Server handle

HonUaHandleDetector prevents usage of an invalid Server handle to the target OPC Server. Currently, a server handle is invalidated if ControlEdge 900 controller resets. A future release of ControlEdge 900 controller will allow OPC UA server handles to ride through a ControlEdge 900 controller reset.

Figure 9-10: HonUaHandleDetector



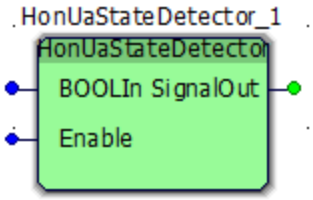
VAR_INPUT		
Enable	BOOL	When set TRUE enables the block functionality. When set FALSE disables the block functionality.
DWORDIn	DWORD	When Enable is set TRUE, the block will monitor DWORDIn for change to 0. If this occurs then SignalOut is set TRUE.

VAR_OUTPUT		
SignalOut	BOOL	See DWORDIn above

Detecting Boolean Resets

HonUaStateDetector is a convenient way to detect that a Boolean flag has been reset from TRUE to FALSE.

Figure 9-11: HonUaStateDetector



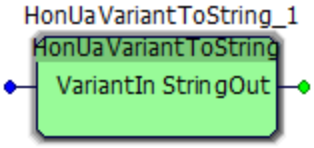
VAR_INPUT		
Enable	BOOL	When set TRUE enables the block functionality. When set FALSE disables the block functionality.
BOOLIn	BOOL	When Enable is set TRUE, the block will monitor BOOLIn for change to FALSE. If this occurs then SignalOut will be set TRUE.

VAR_OUTPUT		
SignalOut	BOOL	See BOOLIn above

Converting Variant Values to String

HonUaVariantToString is useful for debugging purposes. It converts the fields of a variant to a single string.

Figure 9-12: HonUaVariantToString



VAR_INPUT		
VariantIn	UAVariant	Variant value (i.e., as returned from FB “ReadNode”)

VAR_OUTPUT		
StringOut	STRING	String representation of VariantIn

Configuring an OPC UA Client

Binding Protocol to Ethernet Ports

You must establish the physical address or endpoint that enables OPC UA client access to the ControlEdge 900 controller OPC UA Server.

1. From the Home Page of ControlEdge Builder, click the arrow beside **Configure Ethernet Ports**, and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the IP address of the Ethernet port.
3. Under **Protocol Binding**, select **OPC UA Client**.
4. Click **Save** to complete the configuration.

Configuring Parameters for OPC UA Client

OPC UA client maintains sessions in response to each execution of the UaConnect function block. One execution of the UaConnect function block contains that one corresponding session will be created by the OPC UA client on the controller and correspondingly one session will be created on the target OPC UA server.

> **ATTENTION:** Make sure that the OPC UA client's time is synchronized to the controller's time.

To configure an OPC UA client:

1. Click **Configure Protocols** > **OPC UA Client**. The **OPC UA Client** page appears.
2. Select the values for the **Max Session Count** and **Max Subscription Per Session** parameters.

   See the following table for the parameter description.

   Table 9-2: OPC UA Client parameter description

Parameter	Description
Max Session Count	The maximum number of concurrent sessions allowed by the client. If you enter a value of 0, the number of sessions allowed is unlimited.

Parameter	Description
	The default value is 100.
Max Subscriptions Per Session	The maximum number of subscriptions allowed by the client for one session. If you enter a value of 0, the number of subscriptions allowed is unlimited.

3. Click **Save**.

Importing OPC UA Library

To import OPC UA Library

1. In **IEC Programming Workspace**, from the project tree window, right-click **Libraries** and select **Insert> Firmware Library**.

   The Include library dialog appears.

2. Open OPCUA folder and select opcua.fwl, then click **Include**.

The OPCUA library is displayed under **Libraries**.

Importing Data Types of HonUaFbHelperTypes

To import Data Types of HonUaFbHelperTypes

1. In **IEC Programming Workspace**, from the project tree window, select **Data Types**, click **File> Import**.

   The Import / Export dialog appears.

2. Select **Extended IEC 61131-3 Import**, and click **OK**.

   The Object types dialog appears.

3. Select **Datatypes** and click **OK**.

   The Extended IEC 61131-3 dialog appears.

4. Browse to \Users\Public\Documents\ControlEdge Builder\Libraries\OPCUAFBHelpers", and select the target data type. Click **OK**.

The HonUaFbHelperTypes is displayed under **Data Types**.

Importing an OPC UA POU

1. In **IEC Programming Workspace**, from the project tree window, select **Logical POUs**, and click **File> Import**.
   The Import / Export dialog appears.
2. Select **Extended IEC 61131-3 Import**, and click **OK**.
   The Object types dialog appears.
3. Select **POU** and click **OK**.
   The Extended IEC 61131-3 dialog appears.
4. Browse to \Users\Public\Documents\ControlEdge Builder\Libraries\OPCUAFBHelpers", and select the target POU. Click **OK**.

The HonUaFbHelpers is displayed under **Logic POUs**.

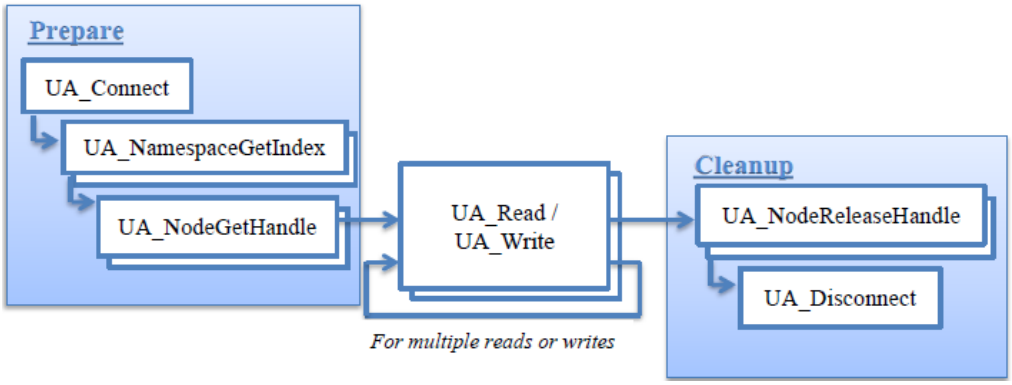
Configuring an OPC UA Logic

ControlEdge PLC OPC UA Client supports data value read and write through below logic sequence.

Make sure that the OPC UA Server's time is synchronized to the Client (the controller's) time.

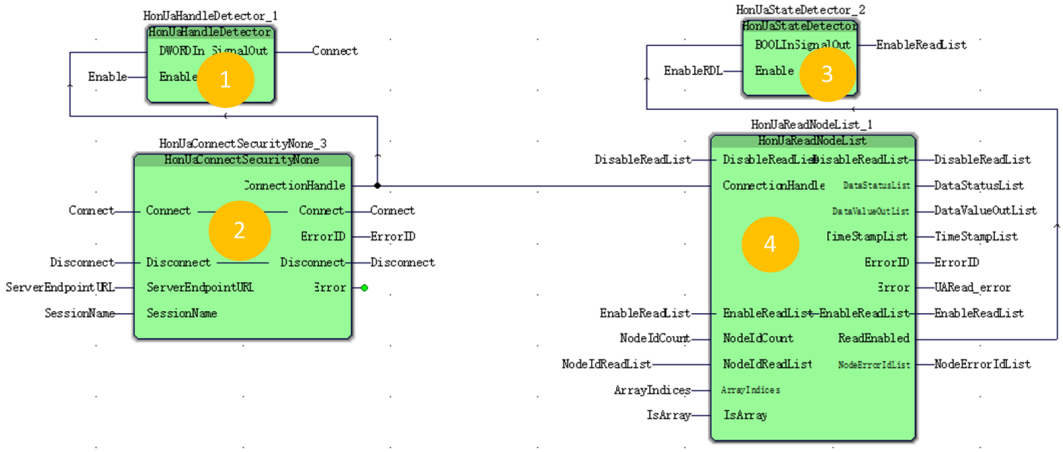
To configure an OPC UA Logic

1. Establish the connection between OPC UA Client and OPC UA Server;
2. Get the namespace index of OPC UA Server once the connection is successfully established;
3. Get the Node handle;
4. Read the value of or write a value to a variable. The following diagram presents the data access workflow:



Example logic for reading list of variables from OPC UA Server

Two main function blocks with some auxiliary logics are used for this application.



1. HonUaConnectSecurityNone – Used for all read/write applications, to establish the connection with OPC UA server or disconnect an existing connection.

   - ServerEndpointURL and sessionName need to be configured
   - Parameter “Disconnect” used to disconnect an existing connection, as there has a auxiliary function block to monitor the status of the connection and handle error scenarions (Enable =1): If no connection is detected (ConnectionHandle=0), this FB 2 will trigger the execution of FB “Ua…” to reestablish the connection.

   Only in two scenarios the value of “ConnectionHandle” will be 0, after power on or disconnect the connection manually, so if user want to disconnect the connection, normally procedure is disable this HonUaHandleDetector first and then do disconnect operation.

2. HonUaReadNodeList

   - DisableReadList – When set TRUE, disables this read block. Read service will not be called with each task cycle.
   - EnableReadList – When set TRUE, enables this read block. Read service will be called with each task cycle. Output of HonUaReadNodeLIst connects to this parameter and the rise edge of this parameter will trigger the execution of HonUaConnectSecurityNone.

- NodeIdCount – The number of Node IDs in NodeIdWriteList. Define how many variable are expected to be read from OPC UA Server.
- NodeIdReadList – Node identifiers of the nodes whose values are to be read by this block (max 20 identifiers).Structure type, consists of three elements

  IdentifierType – Array, to define the identifier type of the variables expected to be read from Server, we can get detail information of the variable in Server side, normally it is string type (Initial value =1);

  Identifier – Array, The variable name

  Namespace – Array, define namespace for each variable, we can get detail information of the variable in Server side or we can use “GetNameSpace”.

OPC UA project sizing and performance

To ensure nominal performance characteristics when executing OPC UA based projects it is recommended that overall memory and CPU usage in the PLC remain less than 50%. This section provides guidelines on factors to consider when enabling OPC UA in ControlEdge 900 controller.

When constructing an OPC UA project there are two factors to consider:

- OPC UA Project Size

  Each project and its related functions require an amount of application data based on its size. If a given project is too big and its application data usage eclipses more than 50% risks instability. The project sizing is measured based on Internal Variables which are directly related to the application data usage within the PLC. More details regarding the sizing of projects is provided in the OPC UA Project Sizing section below.

- OPC UA Performance

  Performance is how the constructed project impacts the PLC. In some cases, it is possible to create a large project that is within the project sizing requirements but requires too many resources to be reliable. The project performance is based on the number of Data Items (e.g. process variables) expected to be read/written

between the OPC UA client and all remote OPC UA servers. Details on how accessing data items affects the PLC are provided in the OPC UA Client Performance, OPC UA Server Performance, OPC UA MDIS Client Performance, and OPC UA MDIS Server Performance sections below.

OPC UA Project Sizing

OPC UA Function Block Instances

OPC UA Function blocks, like all IEC 61131-3 function blocks, add internal variables to the PLC that consume application data memory. The PLC reserves space for application data. To ensure proper functionality, the application data in use should remain at or below 50%. To achieve this, the count of internal variables added to the PLC by all Function block instances should be less than 200,000. There are two methods to estimate the number of internal variables that will be added by a project. The first is less accurate but allows quick estimation using just the total number of data items. The second estimation method can be used during project design phase.

> **TIP:** Maximum 200,000 Internal Variables

- Method #1: Number of Internal Variables Per Data Item

  Depending on the method of data access used by the OPC UA Client, it is possible to calculate the number of internal variables that will be added to the controller. The calculations take into consideration the additional blocks required to support the operations in the table below such as establishing a connection and resolving node IDs. The table below can be used to accurately estimate the internal variable count for projects built to access from 10 to a 1000 data items.

Operations	Internal Variable Per Data Item
Read	165
Write	155
Subscription	1480

> **NOTE:** Reads and writes use fewer internal variables when using ReadList and WriteList blocks versus Read and Write blocks.

- Method #2: Number of Internal Variables Per Function Block

  During the project design phase, the internal variable count can be estimated from the set of function blocks which comprise the project. The tables below identify the number of internal variables required to support the function block. Adding together the internal variable counts for each function block type instance in the project will yield a reasonably accurate count of internal variables for the project.

Helper Block Type	Internal Variable Per Function Block Instance
HonUaCallMethod	1622
HonUaConnectSecurityNone	69
HonUaHandleDetector	31
HonUaReadNode	193
HonUaReadNodeList	3043
HonUaManageSubscription	74
HonUaStateDetector	22
HonUaSubscribeNode	1442
HonUaTranslatePathList	317
HonUaVariantToString	38
HonUaWriteNode	182
HonUaWriteListNode	2858

OPC UA Function Block Type	Internal Variable Per Function Block Instance
UaConnect	37
UaDisconnect	14

OPC UA Function Block Type	Internal Variable Per Function Block Instance
UaMethodCall	539
UaMethodGetHandle	24
UaMethodReleaseHandle	16
UaMonitoredItemAdd	449
UaMonitoredItemRemove	16
UaNamespaceGetIndex	21
UaNodeGetHandle	20
UaNodeGetHandleList	99
UaNodeReleaseHandle	16
UaNodeReleaseHandleList	18
UaRead	104
UaReadList	1781
UaSubscriptionCreate	21
UaSubscriptionDelete	14
UaSubscriptionOperate	21
UaTranslatePath	27
UaTranslatePaths	180
UaWrite	101
UaWriteList	1800

- 100 Data Item Reads and 100 Data Item Writes Example
  - Method #1: Estimating the project size by data item count

    To determine the size of this project we can use the first method to estimate the number of internal variables. This project will have 100 reads and 100 writes, each read on average will use 165 internal variables and each write will use 155 internal variables. This results in an estimated 32,000 internal variables used.

Name of the Function Block	Number of Instances	Internal Variables per Instance	Internal Variables
Reads	100	165	16500
Writes	100	155	15500
Subscriptions	0	1480	0
		Total	32000

- Plan the project layout

  To have the most efficient project design this project will use ReadList and WriteList blocks which support 20 data items at a time. Since we are reading and writing 100 data items, this project will need 5 ReadList blocks and 5 WriteList blocks. All the required function blocks, the required number of instances, and internal variable count are listed in the table below.

- Method #2: Estimating the project size by Function Block

  Estimating the project size by data item can be inaccurate. If the project is close to the limit of 200,000 internal variables it is best to double check the project sizing to ensure the project will be within the limitation. In this case, the project size is so small estimating by function block is not necessary but is done for completion of this guide. Using the table below we can see the calculated internal variable usage will be 29,925.

Name of the Function Block	Number of Instances	Internal Variables per Instance	Internal Variables
HonUaReadNodeList	5	3043	15215
HonUaWriteNodeList	5	2858	14290
HonUaStateDetector	10	22	220
HonUaConnectSecurityNone	2	69	138
HonUaHandleDetector	2	31	62
		Total	29925

- 20 Variable Subscriptions example

- Method #1: Estimating the project size by data item count

  To determine the size of this project by data item count we show that a subscription requires 1480 internal variables per data item. This result in 29600 internal variables used.

Name of the Function Block	Number of Instances	Internal Variables per Instance	Internal Variables
Reads	0	165	0
Writes	0	155	0
Subscriptions	20	1480	29600
		Total	29600

- Plan the project layout

  To create this project, we will need 20 Subscribe blocks, each to read 1 data item. All the required function blocks, the required number of instances, and internal variable count are listed in the table below.

- Method #2: Estimating the project size by function block

  Again, this project is so small that function block estimation is not necessary but is included for the completeness of this guide. From the table below we can see the actual usage is 29,380 internal variables.

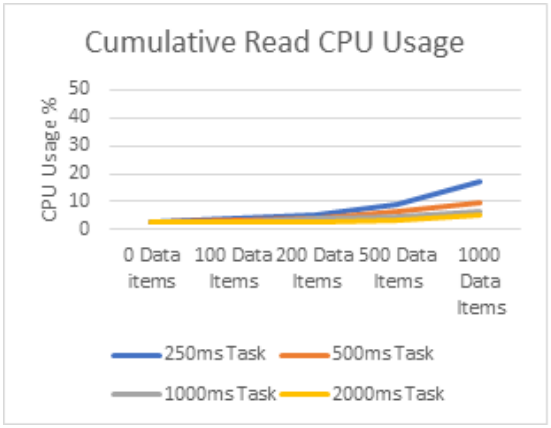
Name of the Function Block	Number of Instances	Internal Variables per Instance	Internal Variables
HonUaSubscribeNode	20	1442	28840
HonUaStateDetector	20	22	440
HonUaConnectSecurityNone	1	69	69
HonUaHandleDetector	1	31	31
		Total	29380

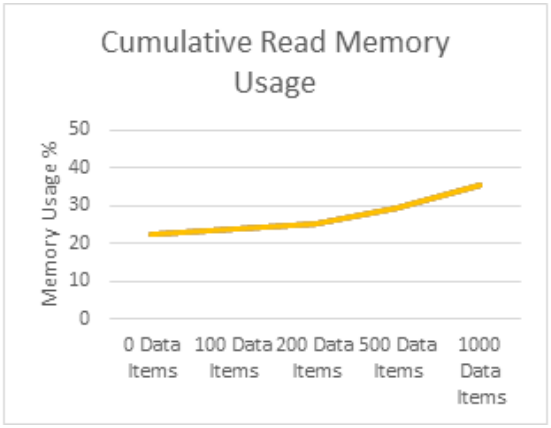
OPC UA Client Performance

Volume of Remote Data Items Accessed and Related Task Interval

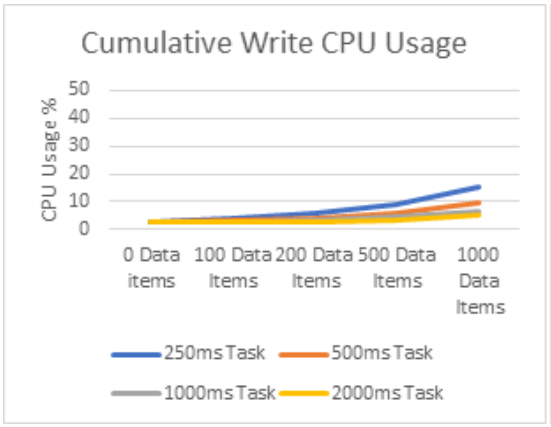
Using the number of data items accessed, we can estimate the performance of the project. In general, the larger the project the larger the impact. Refer to the graphs below which illustrate CPU and RAM usage based on various combinations of variable count and eCLR task interval. CPU usage is heavily influenced by the program task interval. Generally, tasks faster than 250ms should be avoided when building OPC UA projects. As previously stated, maintaining cumulative CPU usage and memory usage at or below 50% will ensure nominal performance.

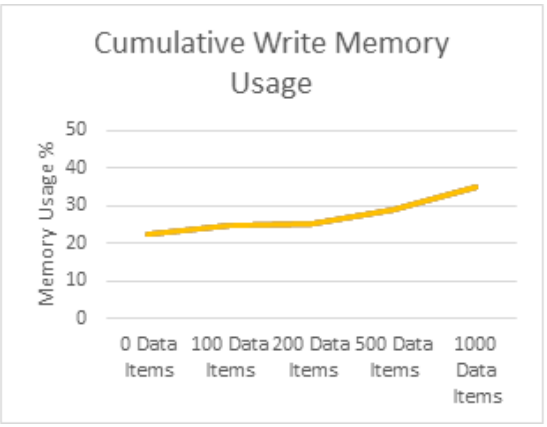
Reads



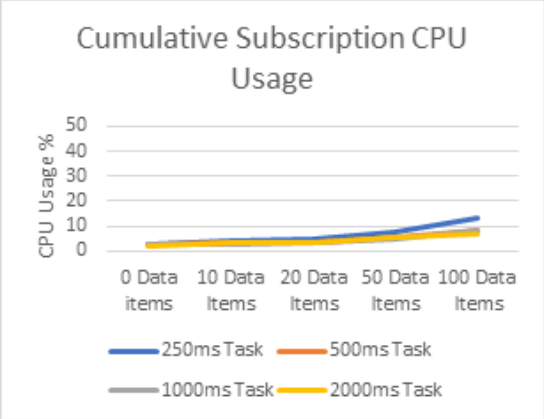


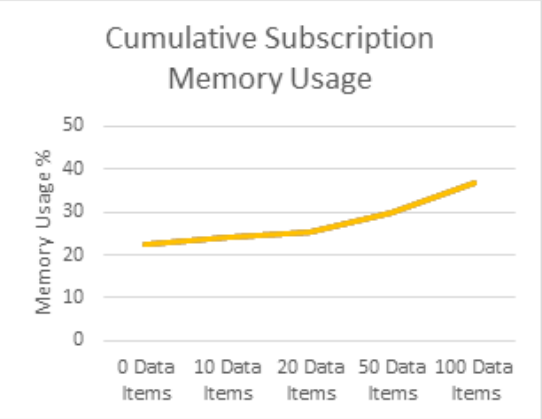
Writes





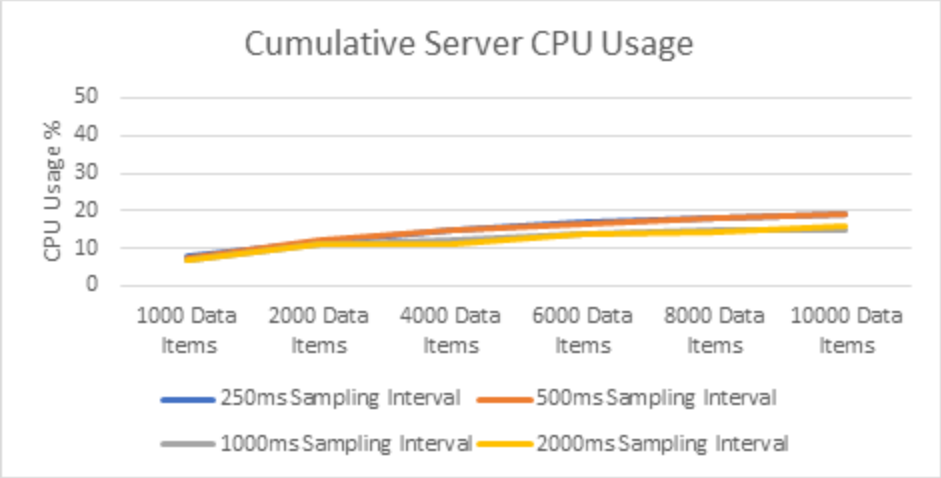
Subscriptions

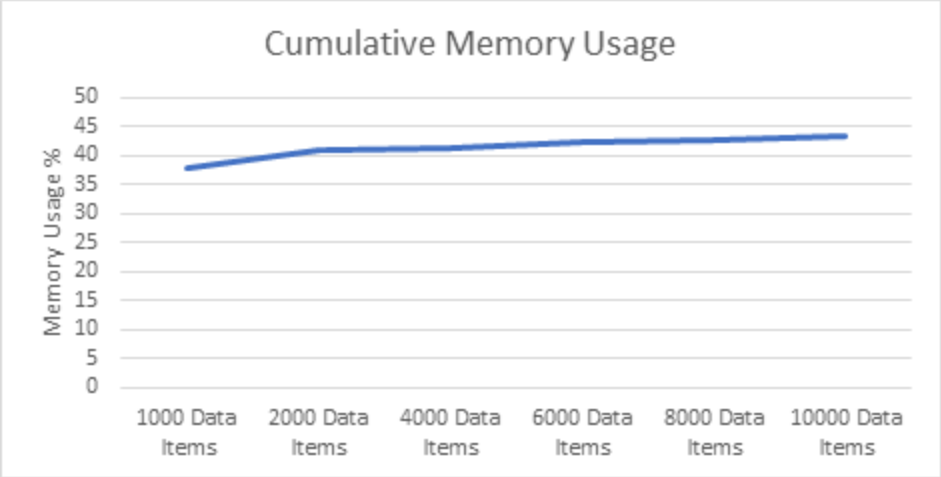




OPC UA Server Performance

The OPC UA server's impact on platform CPU and RAM is primarily defined by external OPC UA client activity. An analysis of this impact at various loading levels is depicted in the graphs below. Each graph includes percent usage based on number of data items and sampling interval. The following data was collected using subscriptions; however, similar results can be expected when executing demand reads and writes at these rates.





Impact of OPC UA MDIS Function Blocks on Project Sizing and Performance

The MDIS OPC UA Object Function blocks are special-purpose blocks that are only used to connect and retrieve data from MDIS OPC UA compliant Servers. The blocks are designed specifically for the MDIS OPC UA information model as defined in the MDIS OPC UA Companion Specification V1.2.

More information on MDIS and the MDIS OPC UA specification can be found here: https://opcfoundation.org/markets-collaboration/mdis/.

MDIS OPC UA Project Sizing

The MDIS Function blocks consume fewer internal variables than the OPC UA function blocks mainly because each Function block knows in advance the parameters and data types of the parameters to be exposed. For example, the MDISInstrObj function block knows that every MDIS Instrument Object has a mandatory variable called “ProcessVariable” of type ‘REAL’. This prior knowledge of the object types allows for more efficient function blocks.

Each OPC UA MDIS function block instance in the project represents a MDIS object within the MDIS server’s address space. To estimate project size, the project engineer must first determine the collection of instruments, valves and other MDIS objects that together represent the target system. Adding together the internal variable counts for each function block type instance in the project will yield a reasonably accurate count of internal variables for the project.

MDIS Block Type	Internal Variables Per Function Block Instance
MDISChokeAbort	16
MDISChokeMove	22
MDISChokeObj	63
MDISChokeSetCalcPos	18
MDISChokeStep	24
MDISDigInstrWriteState	18
MDISDigitalInstrObj	40
MDISDiscreteInstrObj	40
MDISDiscrtLnstrWriteVal	18
MDISInstrObj	56
MDISInstrWriteValue	18
MDISObjEnableDisable	18
MDISValveMove	26
MDISValveObj	58

Varied MDIS Object Example

- Plan the project layout
  - The example below includes a collection of commonly used MDIS function blocks. MDIS function blocks are designed specifically to exchange data between the block and the corresponding MDIS object within the server's address space. Therefore, use of MDIS function blocks requires the use of additional OPC UA function blocks to manage connections, subscriptions and related functionality. A representative collection of related OPC UA function blocks is included in the example below.
- Estimate the project size by function block

The table below calculates the project size to be 2140 Internal Variables

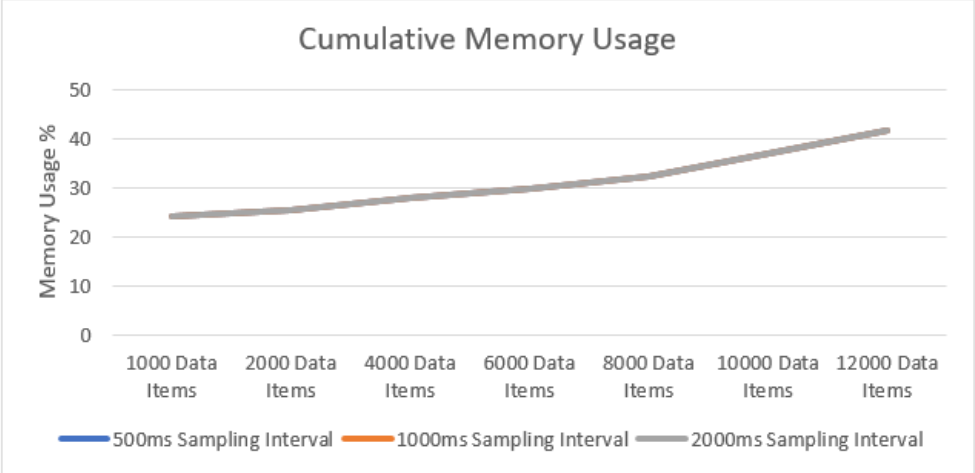
Name of the Function Block	Number of Instances	Internal Variables Per Instance	Internal Variables
MDISInstrObj	15	56	840
MDISDigitalInstrObj	1	40	40
MDISDiscreteInstrObj	2	40	80
MDISValveObj	2	85	170
MDISObjEnableDisable	20	18	360
HonUaConnectSecurityNone	1	69	69
HonUaManageSubscription	1	79	79
HonUaStateDetector	20	22	440
HonUaHandleDetector	2	31	62
		Total	2140

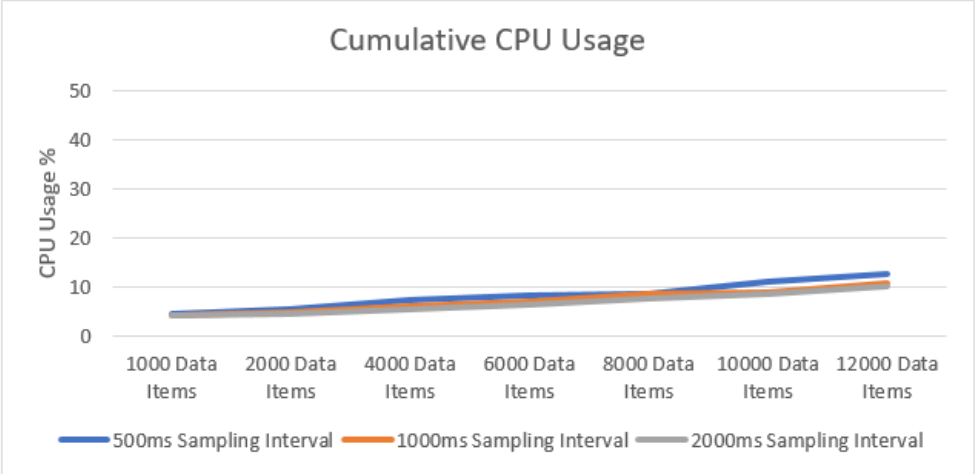
MDIS OPC UA Client Performance

Each MDIS object results in several subscribed data variables in the server. The data items per MDIS object shown in the table below assumes each MDIS object in the server implements all MDIS optional features. The performance impact based on the total number of subscribed data variables is shown in the graphs below.

When estimating the performance impact MDIS will have on the PLC, the total number of data items for a project must calculated. This can be calculated by counting the number of MDIS objects and adding together the total number of data items per object. The resulting data item count can be compared to the following graphs.

MDIS Object Type	Data Items Per MDIS Object
MDISChokeObj	17
MDISDigitalInstrObj	7
MDISDiscreteInstrObj	7
MDISInstrObj	11
MDISValveObj	16





MDIS OPC UA Server Performance

The OPC UA Server handles MDIS objects the same way as all other objects. The resulting performance graphs provided in the OPC UA Server Performance section will remain the same regardless of what function blocks are being used on the OPC UA Client.

OPC UA Error Code Reference

See the following table for OPC UA function block error codes definition:

Error Code	Symbolic ID	Description
16#00000000	success	NA
16#00000001	FB_GEN_ERR_INPUT_PARA_INVALID	The input parameter is invalid.
16#00000002	FB_GEN_ERR_RCV_RSP_TIME_OUT	Time out and no response data is received.
16#00000003	FB_GEN_ERR_INTERNAL_TIME_OUT	IPC is time out.
16#00000004	FB_GEN_ERR_INVALID_REQUEST	The request is invalid.
0x00000000	OpcUa_Good	The operation was successful.
0x80000000	OpcUa_Bad	The operation was unsuccessful but no specific reason is known.
0x80010000	OpcUa_BadUnexpectedError	An unexpected error occurred.
0x80020000	OpcUa_BadInternalError	An internal error occurred as a result of a programming or configuration error.
0x80030000	OpcUa_BadOutOfMemory	Not enough memory to complete the operation.
0x80040000	OpcUa_BadResourceUnavailable	An operating system resource is not

Error Code	Symbolic ID	Description
		available.
0x80050000	OpcUa_BadCommunicationError	A low level communication error occurred.
0x80060000	OpcUa_BadEncodingError	Encoding halted because of invalid data in the objects being serialized.
0x80070000	OpcUa_BadDecodingError	Decoding halted because of invalid data in the stream.
0x80080000	OpcUa_BadEncodingLimitsExceeded	The message encoding/decoding limits imposed by the stack have been exceeded.
0x80B80000	OpcUa_BadRequestTooLarge	The request message size exceeds limits set by the server.
0x80B90000	OpcUa_BadResponseTooLarge	The response message size exceeds limits set by the client.
0x80090000	OpcUa_BadUnknownResponse	An unrecognized response was received from the server.
0x800A0000	OpcUa_BadTimeout	The operation timed out.
0x800B0000	OpcUa_BadServiceUnsupported	The server does not support the requested service.
0x800C0000	OpcUa_BadShutdown	The operation was cancelled because the application is shutting down.
0x800D0000	OpcUa_BadServerNotConnected	The operation could not complete because

Error Code	Symbolic ID	Description
		the client is not connected to the server.
0x800E0000	OpcUa_BadServerHalted	The server has stopped and cannot process any requests.
0x800F0000	OpcUa_BadNothingToDo	There was nothing to do because the client passed a list of operations with no elements.
0x80100000	OpcUa_BadTooManyOperations	The request could not be processed because it specified too many operations.
0x80DB0000	OpcUa_BadTooManyMonitoredItems	The request could not be processed because there are too many monitored items in the subscription.
0x80110000	OpcUa_BadDataTypeIdUnknown	The extension object cannot be (de)serialized because the data type id is not recognized.
0x80120000	OpcUa_BadCertificateInvalid	The certificate provided as a parameter is not valid.
0x80130000	OpcUa_BadSecurityChecksFailed	An error occurred verifying security.
0x80140000	OpcUa_BadCertificateTimeInvalid	The Certificate has expired or is not yet valid.
0x80150000	OpcUa_BadCertificateIssuerTimeInvalid	An Issuer Certificate has expired or is not yet valid.

Error Code	Symbolic ID	Description
0x80160000	OpcUa_BadCertificateHostNameInvalid	The HostName used to connect to a Server does not match a HostName in the Certificate.
d 0x80170000	OpcUa_BadCertificateUriInvali	The URI specified in the ApplicationDescription does not match the URI in the Certificate.
0x80180000	OpcUa_BadCertificateUseNotAllowed	The Certificate may not be used for the requested operation.
0x80190000	OpcUa_BadCertificateIssuerUseNotAllowed	The Issuer Certificate may not be used for the requested operation.
0x801A0000	OpcUa_BadCertificateUntrusted	The Certificate is not trusted.
0x801B0000	OpcUa_BadCertificateRevocationUnknown	It was not possible to determine if the Certificate has been revoked.
0x801C0000	OpcUa_ BadCertificateIssuerRevocationUnknown	It was not possible to determine if the Issuer Certificate has been revoked.
0x801D0000	OpcUa_BadCertificateRevoked	The Certificate has been revoked.
0x801E0000	OpcUa_BadCertificateIssuerRevoked	The Issuer Certificate has been revoked.
0x801F0000	OpcUa_BadUserAccessDenied	User does not have permission to perform the requested operation.
0x80200000	OpcUa_BadIdentityTokenInvalid	The user identity token is not valid.

Error Code	Symbolic ID	Description
0x80210000	OpcUa_BadIdentityTokenRejected	The user identity token is valid but the server has rejected it.
0x80220000	OpcUa_BadSecureChannelIdInvalid	The specified secure channel is no longer valid.
0x80230000	OpcUa_BadInvalidTimestamp	The timestamp is outside the range allowed by the server.
0x80240000	OpcUa_BadNonceInvalid	The nonce does appear to be not a random value or it is not the correct length.
0x80250000	OpcUa_BadSessionIdInvalid	The session id is not valid.
0x80260000	OpcUa_BadSessionClosed	The session was closed by the client.
0x80270000	OpcUa_BadSessionNotActivated	The session cannot be used because ActivateSession has not been called.
0x80280000	OpcUa_BadSubscriptionIdInvalid	The subscription id is not valid.
0x802A0000	OpcUa_BadRequestHeaderInvalid	The header for the request is missing or invalid.
0x802B0000	OpcUa_BadTimestampsToReturnInvalid	The timestamps to return parameter is invalid.
0x802C0000	OpcUa_BadRequestCancelledByClient	The request was cancelled by the client.
0x002D0000	OpcUa_GoodSubscriptionTransferred	The subscription was transferred to another session
0x002E0000	OpcUa_GoodCompletesAsynchronously	The processing will

Error Code	Symbolic ID	Description
		complete asynchronously.
0x002F0000	OpcUa_GoodOverload	Sampling has slowed down due to resource limitations.
0x00300000	OpcUa_GoodClamped	The value written was accepted but was clamped
0x80310000	OpcUa_BadNoCommunication	Communication with the data source is d, but not established, and there is no last known value available.
0x80320000	OpcUa_BadWaitingForInitialData	Waiting for the server to obtain values from the underlying data source.
0x80330000	OpcUa_BadNodeIdInvalid	The syntax of the node id is not valid.
0x80340000	OpcUa_BadNodeIdUnknown	The node id refers to a node that does not exist in the server address space.
0x80350000	OpcUa_BadAttributeIdInvalid	The attribute is not supported for the specified Node.
0x80360000	OpcUa_BadIndexRangeInvalid	The syntax of the index range parameter is invalid.
0x80370000	OpcUa_BadIndexRangeNoData	No data exists within the range of indexes specified.
0x80380000	OpcUa_BadDataEncodingInvalid	The data encoding is invalid.
0x80390000	OpcUa_BadDataEncodingUnsupported	The server does not support the requested

Error Code	Symbolic ID	Description
		data encoding for the node.
0x803A0000	OpcUa_BadNotReadable	The access level does not allow reading or subscribing to the Node.
0x803B0000	OpcUa_BadNotWritable	The access level does not allow writing to the Node.
0x803C0000	OpcUa_BadOutOfRange	The value was out of range.
0x803D0000	OpcUa_BadNotSupported	The requested operation is not supported.
0x803E0000	OpcUa_BadNotFound	A requested item was not found or a search operation ended without success.
0x803F0000	OpcUa_BadObjectDeleted	The object cannot be used because it has been deleted.
0x80400000	OpcUa_BadNotImplemented	Requested operation is not implemented.
0x80410000	OpcUa_BadMonitoringModeInvalid	The monitoring mode is invalid.
0x80420000	OpcUa_BadMonitoredItemIdInvalid	The monitoring item id does not refer to a valid monitored item.
0x80430000	OpcUa_BadMonitoredItemFilterInvalid	The monitored item filter parameter is not valid.
0x80440000	OpcUa_ BadMonitoredItemFilterUnsupported	The server does not support the requested monitored item filter.
0x80450000	OpcUa_BadFilterNotAllowed	A monitoring filter

Error Code	Symbolic ID	Description
		cannot be used in combination with the attribute specified.
0x80460000	OpcUa_BadStructureMissing	A mandatory structured parameter was missing or null.
0x80470000	OpcUa_BadEventFilterInvalid	The event filter is not valid.
0x80480000	OpcUa_BadContentFilterInvalid	The content filter is not valid.
0x80C10000	OpcUa_BadFilterOperatorInvalid	An unregognized operator was provided in a filter.
0x80C20000	OpcUa_BadFilterOperatorUnsupported	A valid operator was provided, but the server does not provide support for this filter operator.
0x80C30000	OpcUa_BadFilterOperandCountMismatch	The number of operands provided for the filter operator was less then expected for the operand provided.
0x80490000	OpcUa_BadFilterOperandInvalid	The operand used in a content filter is not valid.
0x80C40000	OpcUa_BadFilterElementInvalid	The referenced element is not a valid element in the content filter.
0x80C50000	OpcUa_BadFilterLiteralInvalid	The referenced literal is not a valid value.
0x804A0000	OpcUa_BadContinuationPointInvalid	The continuation point provide is longer valid.
0x804B0000	OpcUa_BadNoContinuationPoints	The operation could not be processed

Error Code	Symbolic ID	Description
		because all continuation points have been allocated.
0x804C0000	OpcUa_BadReferenceTypeIdInvalid	The operation could not be processed because all continuation points have been allocated.
0x804D0000	OpcUa_BadBrowseDirectionInvalid	The browse direction is not valid.
0x804E0000	OpcUa_BadNodeNotInView	The node is not part of the view.
0x804F0000	OpcUa_BadServerUriInvalid	The ServerUri is not a valid URI.
0x80500000	OpcUa_BadServerNameMissing	No ServerName was specified.
0x80510000	OpcUa_BadDiscoveryUrlMissing	No DiscoveryUrl was specified.
0x80520000	OpcUa_BadSempahoreFileMissing	The semaphore file specified by the client is not valid.
0x80530000	OpcUa_BadRequestTypeInvalid	The security token request type is not valid.
0x80540000	OpcUa_BadSecurityModeRejected	The security mode does not meet the requirements set by the Server.
0x80550000	OpcUa_BadSecurityPolicyRejected	The security policy does not meet the requirements set by the Server.
0x80560000	OpcUa_BadTooManySessions	The maximum number of sessions has been reached.

Error Code	Symbolic ID	Description
0x80570000	OpcUa_BadUserSignatureInvalid	The user token signature is missing or invalid.
0x80580000	OpcUa_BadApplicationSignatureInvalid	The signature generated with the client certificate is missing or invalid.
0x80590000	OpcUa_BadNoValidCertificates	The client did not provide at least one software certificate that is valid and meets the profile requirements for the server.
0x80C60000	OpcUa_BadIdentityChangeNotSupported	The Server does not support changing the user identity assigned to the session.
0x805A0000	OpcUa_BadRequestCancelledByRequest	The request was cancelled by the client with the Cancel service.
0x805B0000	OpcUa_BadParentNodeIdInvalid	The parent node id does not to refer to a valid node.
0x805C0000	OpcUa_BadReferenceNotAllowed	The reference could not be created because it violates constraints imposed by the data model.
0x805D0000	OpcUa_BadNodeIdRejected	The requested node id was reject because it was either invalid or server does not allow node ids to be specified by the client.
0x805E0000	OpcUa_BadNodeIdExists	The requested node id is already used by

Error Code	Symbolic ID	Description
		another node.
0x805F0000	OpcUa_BadNodeClassInvalid	The node class is not valid.
0x80600000	OpcUa_BadBrowseNameInvalid	The browse name is invalid.
0x80610000	OpcUa_BadBrowseNameDuplicated	The browse name is not unique among nodes that share the same relationship with the parent.
0x80620000	OpcUa_BadNodeAttributesInvalid	The node attributes are not valid for the node class.
0x80630000	OpcUa_BadTypeDefinitionInvalid	The type definition node id does not reference an appropriate type node.
0x80640000	OpcUa_BadSourceNodeIdInvalid	The source node id does not reference a valid node.
0x80650000	OpcUa_BadTargetNodeIdInvalid	The target node id does not reference a valid node.
0x80660000	OpcUa_BadDuplicateReferenceNotAllowed	The reference type between the nodes is already d.
0x80670000	OpcUa_BadInvalidSelfReference	The server does not allow this type of self reference on this node.
0x80680000	OpcUa_BadReferenceLocalOnly	The reference type is not valid for a reference to a remote server.
0x80690000	OpcUa_BadNoDeleteRights	The server will not allow the node to be deleted.

Error Code	Symbolic ID	Description
0x40BC0000	OpcUa_UncertainReferenceNotDeleted	The server was not able to delete all target references.
0x806A0000	OpcUa_BadServerIndexInvalid	The server index is not valid.
0x806B0000	OpcUa_BadViewIdUnknown	The view id does not refer to a valid view node.
0x80C90000	OpcUa_BadViewTimestampInvalid	The view timestamp is not available or not supported.
0x80CA0000	OpcUa_BadViewParameterMismatch	The view parameters are not consistent with each other.
0x80CB0000	OpcUa_BadViewVersionInvalid	The view version is not available or not supported.
0x40C00000	OpcUa_UncertainNotAllNodesAvailable	The list of references may not be complete because the underlying system is not available.
0x00BA0000	OpcUa_GoodResultsMayBeIncomplete	The server should have followed a reference to a node in a remote server but did not. The result set may be incomplete.
0x80C80000	OpcUa_BadNotTypeDefinition	The provided Nodeid was not a type definition nodeid.
0x406C0000	OpcUa_UncertainReferenceOutOfServer	One of the references to follow in the relative path references to a node in the address space in another server.

Error Code	Symbolic ID	Description
0x806D0000	OpcUa_BadTooManyMatches	The requested operation has too many matches to return.
0x806E0000	OpcUa_BadQueryTooComplex	The requested operation requires too many resources in the server.
0x806F0000	OpcUa_BadNoMatch	The requested operation has no match to return.
0x80700000	OpcUa_BadMaxAgeInvalid	The max age parameter is invalid.
0x80710000	OpcUa_BadHistoryOperationInvalid	The history details parameter is not valid.
0x80720000	OpcUa_BadHistoryOperationUnsupported	The server does not support the requested operation.
0x80BD0000	OpcUa_BadInvalidTimestampArgument	The d timestamp to return was invalid.
0x80730000	OpcUa_BadWriteNotSupported	The server not does support writing the combination of value, status and timestamps provided.
0x80740000	OpcUa_BadTypeMismatch	The value supplied for the attribute is not of the same type as the attribute's value.
0x80750000	OpcUa_BadMethodInvalid	The method id does not refer to a method for the specified object.
0x80760000	OpcUa_BadArgumentsMissing	The client did not specify all of the input arguments for the

Error Code	Symbolic ID	Description
		method.
0x80770000	OpcUa_BadTooManySubscriptions	The server has reached its maximum number of subscriptions.
0x80780000	OpcUa_BadTooManyPublishRequests	The server has reached the maximum number of queued publish requests.
0x80790000	OpcUa_BadNoSubscription	There is no subscription available for this session.
0x807A0000	OpcUa_BadSequenceNumberUnknown	The sequence number is unknown to the server.
0x807B0000	OpcUa_BadMessageNotAvailable	The requested notification message is no longer available.
0x807C0000	OpcUa_BadInsufficientClientProfile	The Client of the current Session does not support one or more Profiles that are necessary for the Subscription.
0x80BF0000	OpcUa_BadStateNotActive	The sub-state machine is not currently active.
0x807D0000	OpcUa_BadTcpServerTooBusy	The server cannot process the request because it is too busy.
0x807E0000	OpcUa_BadTcpMessageTypeInvalid	The type of the message specified in the header invalid.
0x807F0000	OpcUa_BadTcpSecureChannelUnknown	The SecureChannelId and/or TokenId are not currently in use.
0x80800000	OpcUa_BadTcpMessageTooLarge	The size of the message specified in

Error Code	Symbolic ID	Description
		the header is too large.
0x80810000	OpcUa_BadTcpNotEnoughResources	There are not enough resources to process the request.
0x80820000	OpcUa_BadTcpInternalError	An internal error occurred.
0x80830000	OpcUa_BadTcpEndpointUrlInvalid	The Server does not recognize the QueryString specified.
0x80840000	OpcUa_BadRequestInterrupted	The request could not be sent because of a network interruption.
0x80850000	OpcUa_BadRequestTimeout	Timeout occurred while processing the request.
0x80860000	OpcUa_BadSecureChannelClosed	The secure channel has been closed.
0x80870000	OpcUa_BadSecureChannelTokenUnknown	The token has expired or is not recognized.
0x80880000	OpcUa_BadSequenceNumberInvalid	The sequence number is not valid.
0x80BE0000	OpcUa_BadProtocolVersionUnsupported	The applications do not have compatible protocol versions.
0x80890000	OpcUa_BadConfigurationError	There is a problem with the configuration that affects the usefulness of the value.
0x808A0000	OpcUa_BadNotConnected	The variable should receive its value from another variable, but has never been configured to do so.
0x808B0000	OpcUa_BadDeviceFailure	There has been a

Error Code	Symbolic ID	Description
		failure in the device/data source that generates the value that has affected the value.
0x808C0000	OpcUa_BadSensorFailure	There has been a failure in the sensor from which the value is derived by the device/data source.
0x808D0000	OpcUa_BadOutOfService	The source of the data is not operational.
0x808E0000	OpcUa_BadDeadbandFilterInvalid	The deadband filter is not valid.
0x408F0000	OpcUa_ UncertainNoCommunicationLastUsableValue	Communication to the data source has failed. The variable value is the last value that had a good quality.
0x40900000	OpcUa_UncertainLastUsableValue	Whatever was updating this value has stopped doing so.
0x40910000	OpcUa_UncertainSubstituteValue	The value is an operational value that was manually overwritten.
0x40920000	OpcUa_UncertainInitialValue	The value is an initial value for a variable that normally receives its value from another variable.
0x40930000	OpcUa_UncertainSensorNotAccurate	The value is at one of the sensor limits.
0x40940000	OpcUa_ UncertainEngineeringUnitsExceeded	The value is outside of the range of values d for this parameter.
0x40950000	OpcUa_UncertainSubNormal	The value is derived

Error Code	Symbolic ID	Description
		from multiple sources and has less than the required number of Good sources.
0x00960000	OpcUa_GoodLocalOverride	The value has been overridden.
0x80970000	OpcUa_BadRefreshInProgress	This Condition refresh failed, a Condition refresh operation is already in progress.
0x80980000	OpcUa_BadConditionAlreadyDisabled	This condition has already been disabled.
0x80CC0000	OpcUa_BadConditionAlreadyEnabled	This condition has already been enabled.
0x80990000	OpcUa_BadConditionDisabled	Property not available, this condition is disabled.
0x809A0000	OpcUa_BadEventIdUnknown	The specified event id is not recognized.
0x80BB0000	OpcUa_BadEventNotAcknowledgeable	The event cannot be acknowledged.
0x80CD0000	OpcUa_BadDialogNotActive	The dialog condition is not active.
0x80CE0000	OpcUa_BadDialogResponseInvalid	The response is not valid for the dialog.
0x80CF0000	OpcUa_BadConditionBranchAlreadyAcked	The condition branch has already been acknowledged.
0x80D00000	OpcUa_ BadConditionBranchAlreadyConfirmed	The condition branch has already been confirmed.
0x80D10000	OpcUa_BadConditionAlreadyShelved	The condition has already been shelved.
0x80D20000	OpcUa_BadConditionNotShelved	The condition is not currently shelved.

Error Code	Symbolic ID	Description
0x80D30000	OpcUa_BadShelvingTimeOutOfRange	The shelving time not within an acceptable range.
0x809B0000	OpcUa_BadNoData	No data exists for the requested time range or event filter.
0x80D70000	OpcUa_BadBoundNotFound	No data found to provide upper or lower bound value.
0x80D80000	OpcUa_BadBoundNotSupported	The server cannot retrieve a bound for the variable.
0x809D0000	OpcUa_BadDataLost	Data is missing due to collection started/stopped/lost.
0x809E0000	OpcUa_BadDataUnavailable	Expected data is unavailable for the requested time range due to an un-mounted volume, an off-line archive or tape, or similar reason for temporary unavailability.
0x809F0000	OpcUa_BadEntryExists	The data or event was not successfully inserted because a matching entry exists.
0x80A00000	OpcUa_BadNoEntryExists	The data or event was not successfully updated because no matching entry exists.
0x80A10000	OpcUa_BadTimestampNotSupported	The client requested history using a timestamp format the server does not support (i.e requested ServerTimestamp

Error Code	Symbolic ID	Description
		when server only supports SourceTimestamp).
0x00A20000	OpcUa_GoodEntryInserted	The data or event was successfully inserted into the historical database.
0x00A30000	OpcUa_GoodEntryReplaced	The data or event field was successfully replaced in the historical database.
0x40A40000	OpcUa_UncertainDataSubNormal	The value is derived from multiple values and has less than the required number of Good values.
0x00A50000	OpcUa_GoodNoData	No data exists for the requested time range or event filter.
0x00A60000	OpcUa_GoodMoreData	The data or event field was successfully replaced in the historical database.
0x80D40000	OpcUa_BadAggregateListMismatch	The requested number of Aggregates does not match the requested number of NodeIds.
0x80D50000	OpcUa_BadAggregateNotSupported	The requested Aggregate is not support by the server.
0x80D60000	OpcUa_BadAggregateInvalidInputs	The aggregate value could not be derived due to invalid data inputs.
0x80DA0000	OpcUa_BadAggregateConfigurationRejected	The aggregate configuration is not

Error Code	Symbolic ID	Description
		valid for specified node.
0x00D90000	OpcUa_GoodDataIgnored	The request pecifies fields which are not valid for the EventType or cannot be saved by the historian.
0x00A70000	OpcUa_GoodCommunicationEvent	The communication layer has raised an event.
0x00A80000	OpcUa_GoodShutdownEvent	The system is shutting down.
0x00A90000	OpcUa_GoodCallAgain	The operation is not finished and needs to be called again.
0x00AA0000	OpcUa_GoodNonCriticalTimeout	A non-critical timeout occurred.
0x80AB0000	OpcUa_BadInvalidArgument	One or more arguments are invalid.
0x80AC0000	OpcUa_BadConnectionRejected	Could not establish a network connection to remote server.
0x80AD0000	OpcUa_BadDisconnect	The server has disconnected from the client.
0x80AE0000	OpcUa_BadConnectionClosed	The network connection has been closed.
0x80AF0000	OpcUa_BadInvalidState	The operation cannot be completed because the object is closed, uninitialized or in some other invalid state.
0x80B00000	OpcUa_BadEndOfStream	Cannot move beyond end of the stream.

Error Code	Symbolic ID	Description
0x80B10000	OpcUa_BadNoDataAvailable	No data is currently available for reading from a non-blocking stream.
0x80B20000	OpcUa_BadWaitingForResponse	The asynchronous operation is waiting for a response.
0x80B30000	OpcUa_BadOperationAbandoned	The asynchronous operation was abandoned by the caller.
0x80B40000	OpcUa_BadExpectedStreamToBlock	The stream did not return all data requested (possibly because it is a non-blocking stream).
0x80B50000	OpcUa_BadWouldBlock	Non blocking behaviour is required and the operation would block.
0x80B60000	OpcUa_BadSyntaxError	A value had an invalid syntax.
0x81000000	OpcUa_StartOfStackStatusCodes	Begin of status codes internal to the stack.
0x81010000	OpcUa_BadSignatureInvalid	The message signature is invalid.
0x81040000	OpcUa_BadExtensibleParameterInvalid	The extensible parameter provided is not a valid for the service.
0x81050000	OpcUa_ BadExtensibleParameterUnsupported	The extensible parameter provided is valid but the server does not support it.
0x81060000	OpcUa_BadHostUnknown	The hostname could not be resolved.

Error Code	Symbolic ID	Description
0x81070000	OpcUa_BadTooManyPosts	Too many posts were made to a semaphore.
0x81080000	OpcUa_BadSecurityConfig	The security configuration is not valid.
0x81090000	OpcUa_BadFileNotFound	Invalid file name specified.
0x810A0000	OpcUa_BadContinue	Accept bad result and continue anyway.
0x810B0000	OpcUa_BadHttpMethodNotAllowed	Accept bad result and continue anyway.
0x810C0000	OpcUa_BadFileExists	File exists.

CHAPTER

10 CDA CONFIGURATION

CDA is short for Control Data Access. CDA protocol supports peer to peer communication between ControlEdge 900 controller with C300 controller or ACE or SIM-300 or SIM-ACE or UOC. ControlEdge 900 controller acts as the CDA responder and C300 controller (or the others mentioned above) acts as the CDA initiator.

It supports:

- Maximum 20 CDA initiators connected to a single ControlEdge 900 CPM
- Maximum 1000 PPS (parameters per second) between CDA initiators and ControlEdge 900 CPM
- Both read and write access from C300 or ACE or UOC controller
- Read access from SIM-300 or SIM-ACE
- Communication Security including IPsec and embedded Firewall

To configure a CDA responder, perform the following steps:

In this section:

Installing ControlEdge integration service

Starting with Experion R501.1, you can communicate with the following controllers in the Experion PKS system through CDA. You should install and start the ControlEdge integration service on the Experion Server.

- C300
- ACE
- Sim-C300
- Sim-ACE

> **ATTENTION:** It is required to install the ControlEdge integration service on both Experion servers when using Experion Server redundancy, and all Server nodes in the Experion Backup Conotrol Center topology.

To install the ControlEdge integration service

1. Insert the **ControlEdge Builder Media Kit** into the DVD-ROM drive.
2. Browse to the folder ControlEdgeIntegrationService, and double-click the file **ControlEdgeIntegrationService.exe**.
3. The **ControlEdgeIntegrationService - InstallShied** Wizard dialog appears. Click Next.
4. In the **License Agreement** page, click **I accept the terms in the license agreement** and click **Next**.
5. In the **ExpAcctSvcLP Login** page, enter the **Username**, **Password** and **Confirm password** for the user account that the ControlEdge Integration Service shall log on as. Click **Next**.

   > **ATTENTION:** The user name must be started with ".\". The user should have a "Security level" of at least "Engineer" in the Experion server. See "Configuring system security" in the *Experion Server and Client Configuration Guide* for more information.

6. In the **Setup Type** page, select the setup type that best suits your needs. It is recommended to select **Complete**. Click **Next**.
7. In the **Ready to Install the Program** page, click **Install** to begin the installation. You can click **Cancel** to abort the installation.

8. The installation is in progress.
9. The **InstallShield Wizard Completed** dialog appears. Click **Finish**.

To check the status of the ControlEdge integration service

1. Click **Start button** of PC, and enter **services.msc** in the search bar. The **Services** dialog appears.
2. Find **Honeywell ControlEdge Integration Service**, and ensure the **Status** is **Running**. If not, right-click the service and click **Start**.
3. Check the **Startup Type** is **Automatic**. If not, right-click the service and select **Properties**, and then select **Automatic** from the **Startup type** drop-down list.

Configuring a CDA Responder

A new project is created and a controller is added to the project in ControlEdge Builder. See "Creating a project" and "Connecting a controller" in *ControlEdge Builder User's Guide* for more details.

To set a controller as a CDA responder

1. From the Home Page, click **Configure Ethernet Ports** and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the details in the **IP Address**, **Subnet Mask** and **Gateway** fields.

   > **TIP:** The IP addresses for the controller and Experion devices to be communicated must be on the same subnet.

3. Under **Protocol Binding**, select **CDA Responder** to bind CDA responder to the Ethernet port.
4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.
5. This step ONLY applies to projects with versions prior to R161.

   Select CDA from the global variables or local variables you want to publish to Experion.

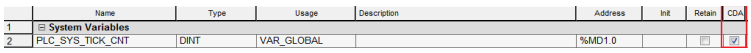
   See the following table for the data type matching between the ControlEdge 900 controller variables and Experion Server parameters.

Data type in ControlEdge 900 controller	Data type in Experion
IEC_BOOL	BOOLEAN
IEC_SINT	INT8
IEC_INT	INT16
IEC_DINT	INT32
IEC_USINT	UINIT8
IEC_UINT	UINT16
IEC_UDINT	UINT32
IEC_REAL	FLOAT32
IEC_LREAL	FLOAT64
IEC_BYTE	UINT8
IEC_WORD	UINT16
IEC_DWORD	UINT32
IEC_ULINT	UINT64
IEC_LWORD	UINT64
IEC_STRING	STRING
IEC_STRUCT	See Note 1 below.
Note 1: Structure is a data type of I/O variable, so you should create a single variable for each parameter in the structure for CDA communication.	

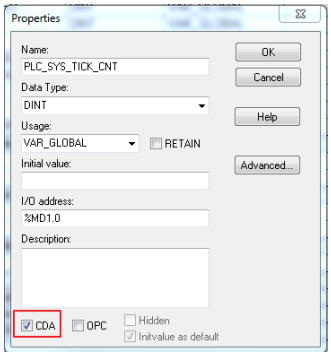
a. Click **IEC Programming Workspace** from the toolbar.

b. Perform either of the following methods to select CDA.

- From the variable sheets, select CDA.



- From the variable properties dialog, select CDA.



6. Click **Make** to compile the configuration to the controller.

Publishing to Experion

For peer to peer connection to C300 or other Experion CEE based controller, you need to publish the configuration to the Experion Server through CDA.

Publishing when ControlEdge Builder is launched from Configuration Studio

In this scenario, ControlEdge Builder is launched from **Configuration Studio** > **Control Strategy** > **ControlEdge Integration** > **Configure control strategy**, the ControlEdge Builder is already running under the user credentials supplied from Configuration Studio.

From the Home Page, click **Publish to Experion** under **Programming and I/O**. The project configuration will be published to Experion directly.

Publishing when ControlEdge Builder is launched separately on an Experion node

In this scenario the Experion client components are available on the node, but the context for establishing the connection to the Server and the user credentials are not available from Configuration Studio.

Prerequisites

ControlEdge Builder is launched separately on an Experion node, not from Configuration Studio.

1. From the Home Page, click **Publish to Experion** under **Programming and I/O**.
2. Enter **Experion server**, **Domain**, **User name** and **Password**.
   See the following table for the parameter description.

Parameter	Description
Experion server	The base part of the Experion server name you would like to connect to. For example if you have redundant Experion Servers, enter the hostname of the server without the last “A” or “B” letter.
Use current Windows account	Connect with current Windows account. If you select this checkbox, you do not need to enter **Domain**, **User name** or **Password**.
Domain	Domain name. If you enter ".", it means current application domain.
User name	User account name
Password	Password for the user account, which is case-sensitive.

3. Click **Publish**. A message appears indicating that the configuration is published successfully. Click **OK**.

Publishing when ControlEdge Builder is launched on non-Experion node

The “Experion client components” optional installation package must be installed from the Experion Installation Media which you want to communicate with for the **Publish to Experion** function to work. These can be installed through the “Optional Components” selection on the installation media. If these components are not installed, a message appears indicating “Unable to publish to Experion as client components are not installed”, but all other ControlEdge Builder functions should continue to work as expected.

Prerequisites

- The "Experion client components" are installed on the ControlEdge Builder node.

- ControlEdge Builder is installed on a same version of Microsoft Windows that is supported for either Experion Client or Server that you want to communicate with. Refer to the Experion specifications for the specific release for supported operation system details.
- ControlEdge Builder is launched.

1. From the Home Page, click **Publish to Experion** under **Programming and I/O**.
2. Enter **Experion server**, **Domain**, **User name** and **Password**.
   See the following table for the parameter description.

Parameter	Description
Experion server	The base part of the Experion server name you would like to connect to. For example if you have redundant Experion Servers, enter the hostname of the server without the last “A” or “B” letter.
Use current Windows account	Connect with current Windows account. If you select this checkbox, you do not need to enter **Domain**, **User name** or **Password**.
Domain	Domain name. If you enter ".", it means current application domain.
User name	User account name
Password	Password for the user account, which is case-sensitive.

3. Click **Publish**. A message appears indicating that the configuration is published successfully. Click **OK**.

CHAPTER

11 MQTT CONFIGURATION

MQTT (Message Queuing Telemetry Transport) is an open OASIS and ISO standard (ISO/IEC 20922) lightweight, publish-subscribe network protocol that transports messages between devices. The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections.

Controllers support MQTT messaging with Sparkplug B payloads to communicate with SCADA/IIOT Host since R170.

Configuring MQTT

1. From the Home Page, click **Configure Ethernet Ports** and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the details in the **IP Address**, **Subnet Mask** and **Gateway** fields.
3. Under **Protocol Binding**, select **MQTT** to bind MQTT to the Ethernet port.
4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.
5. Under **I/O and Communications** tab, click **Configure Protocols** > **MQTT**.
6. Click **Add Connection**, and the **Add MQTT Connection** dialog appears.
7. Select **Ethernet port**.
8. Click **OK** to add MQTT connection.
9. In the **Basic Configuration** group, configure the following parameters.

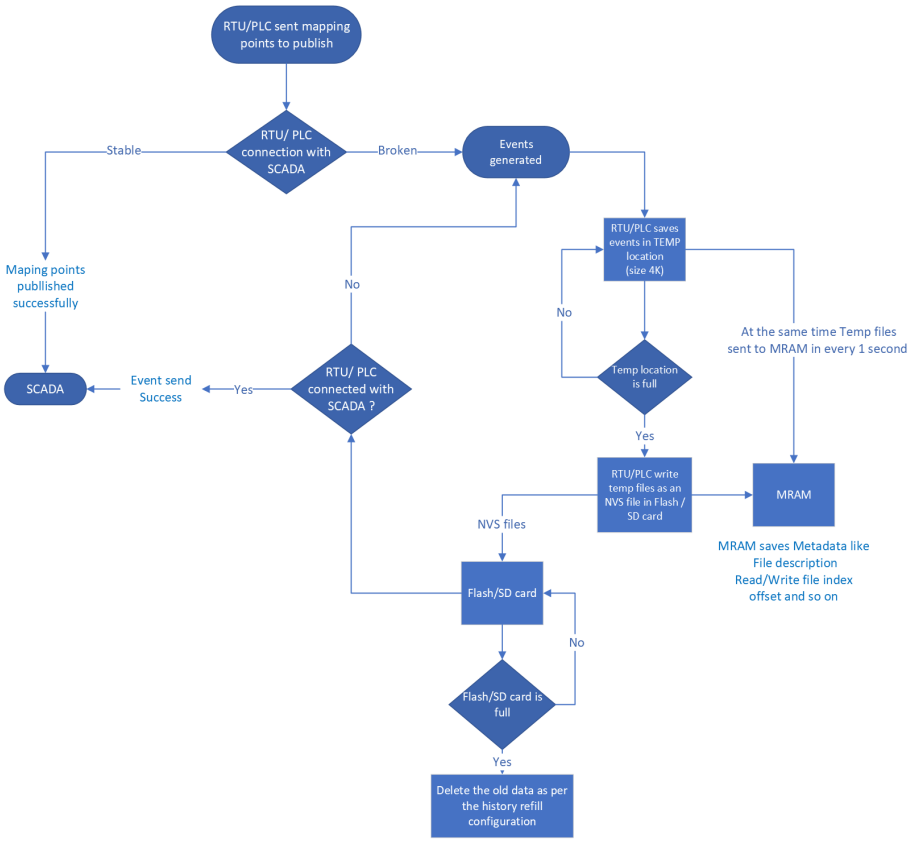
Parameter	Description
Broker Type	Select the broker type from the drop down list. 1. URL 2. IPv4

Parameter	Description
Broker Address	Based on the selected Broker type enter the address. 1. IPv4 : Enter broker IP address. 2. URL: Enter broker Domain name.
TCP Port	The TCP port of MQTT broker.
More	Click **More**, the **Broker List** dialog appears. You can add/delete one or more MQTT brokers. For each broker, you can edit **Broker Address** and **TCP Port**. By default, the controller establishes the connection with the first broker in the **Broker List**. If the external SCADA cannot be accessed via the current broker, the controller will switch to another broker and recover the connection. Up to two brokers can be added in the **Broker List**. **NOTE:** If you want to reorder the broker list, you need to delete brokers and re-add them in order.
Clean Session	Specifies the handling of the Session state. When this option is checked, the controller will not receive old Application Messages and has to subscribe afresh to any topics that it is interested in each time it connects. When this option is unchecked, the controller will receive all QoS 1 or QoS 2 messages that were published while it was disconnected. Hence, to ensure that you do not lose messages while disconnected, use QoS 1 or QoS 2 with CleanSession unchecked.
Keep alive interval	A time interval measured in seconds. Expressed as a 16-bit word, it is the maximum time interval that is permitted to elapse between the point at which the controller finishes transmitting one Control Packet and the point it starts sending the next.
Enable TLS	Used to enable/disable TLS security for MQTT.

Parameter	Description
	If this option is enabled, you should configure the CA certificate. For more information, see "Updating Trust Chain" in *ControlEdge Builder User's Guide*.
Enable CRL	Used to enable/disable CRL function. **NOTE:** It is required to enable TLS if CRL is enabled. If this option is enabled, you must configure CRL. For more information, see "Updating Certificate Revocation List" in *ControlEdge Builder User's Guide*.
Connection Timeout	Allowed maximum waiting time for the establishing of network connection between the controller and MQTT broker.
Client ID	Identifier of the controller. The Client ID should be maximum 23 UTF-8 encoded bytes in length, and contains only the characters 0~9, a~z and A~Z.
SCADA Host ID	Identifier of SCADA/IIoT Host. The SCADA Host ID should be maximum 23 UTF-8 encoded bytes in length, and contains alphanumeric characters with the exception of the reserved characters of '+' (plus), '/' (forward slash), and '#' (number sign).
Group ID	Identifier of logical grouping of MQTT EoN nodes. The Group ID should be maximum 23 UTF-8 encoded bytes in length, and contains alphanumeric characters with the exception of the reserved characters of '+' (plus), '/' (forward slash), and '#' (number sign).
Node ID	Identifier of MQTT EoN node. The Node ID should be maximum 23 UTF-8 encoded bytes in length, and contains alphanumeric characters with the exception of the reserved characters of '+' (plus), '/' (forward slash), and '#' (number sign).

Configure MQTT Store & Forward

The MQTT Store & Forward feature helps users to save the mapping points as events when the controller loses communication with the SCADA, which can then be forwarded along with the live data when communication is restored. See the following figure to understand the workflow.



10. Prepare SD card to store the events. See Preparing SD card.
11. Click **Save MQTT Events to** and select **Flash** or **SD card** from the drop down list.
12. Click **Store & Forward** to display more configuration options.
    a. Enable the **Store and Forward** feature, and configure the following parameters.

Parameter	Description
Delete	Used to enable/disable the deletion of the oldest

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>Oldest Events on Events Overflow</td><td>events on SD card or Flash card overflow storage

Enable: When a new event arrives, the oldest event gets deleted when events overflow on SD card or Flash card.

Disable: No new events stored when events overflow on SD card or Flash card.</td></tr>
<tr><td>Maximum Events to Send</td><td>Configure the maximum events to send to SCADA on every 100 milliseconds.

NOTE: By default, the maximum events to send is configured as 60 events on every 100 milliseconds.

The configurable range for the Maximum events to send is:

• Maximum 300 events on every 100 milliseconds.

• Minimum 10 events on every 100 milliseconds.

The following recommended values matrix for different bandwidths:

<table>
<tr><th>Max Events to send</th><th>Packages per Second</th><th>Event Size (Bytes)</th><th>Theory Bandwidth (Kbps)</th></tr>
<tr><td>10</td><td>10</td><td>32</td><td>25</td></tr>
<tr><td>20</td><td>10</td><td>32</td><td>50</td></tr>
<tr><td>30</td><td>10</td><td>32</td><td>75</td></tr>
<tr><td>40</td><td>10</td><td>32</td><td>100</td></tr>
<tr><td>50</td><td>10</td><td>32</td><td>125</td></tr>
<tr><td>60</td><td>10</td><td>32</td><td>150</td></tr>
</table>
</td></tr>
</table>

Parameter	Description

Max Events to send	Packages per Second	Event Size (Bytes)	Theory Bandwidth (Kbps)
70	10	32	175
80	10	32	200
90	10	32	225
100	10	32	250
150	10	32	375
200	10	32	500
250	10	32	625
300	10	32	750

See Configure the MQTT Store & Forward feature on the Experion Quick Builder .

See MQTT Diagnostics for more information.

13. Click **Publish** to display more configuration options, and configure the following parameters.

Parameter	Description
Topic	Topic is part of a MQTT message. All MQTT clients using the Sparkplug™ specification will use the following Topic Namespace structure: *namespace/'Group ID'/message_type/'Node ID'*.
QoS	Configure the Quality of Service level of the data topic. There are 3 QoS levels in MQTT: • At most once delivery (0) • At least once delivery (1) • Exactly once delivery (2)
Payload	Select MQTT mapping from the drop-down list. For more information, see "Adding a MQTT mapping

Parameter	Description
	table" in *ControlEdge Builder User's Guide*.
Trigger Type	Select the trigger type. Event: Publish data when data changes. Periodic: Publish data periodically.
Interval (seconds)	The time interval is measured in seconds, and the default value is 30. It is only configurable when **TriggerType** is set to Periodic.

14. In the **Subscribe** group, specify the **QoS** level of the subscribe topics.
15. Click **Save**.

16. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
17. Click **Download** from the Home Page to load the configuration of MQTT to the controller.

CHAPTER

12 IEC60870-5-104 OUTSTATION CONFIGURATION

ControlEdge 2020 and ControlEdge 900 Controllers, as an IEC60870-5-104 Outstation, support IEC60870-5 SCADA communication through Ethernet.

Configuring IEC60870-5-104 Outstation

1. From the Home Page, click **Configure Ethernet Ports** and select **ETH1** or **ETH2**.
2. Under **Network Setting**, select **Use the following IP address** and enter the details in the **IP Address**, **Subnet Mask** and **Gateway** fields.
3. Under **Protocol Binding**, select **IEC60870-5-104 Outstation** to bind IEC60870-5-104 Outstation to the Ethernet port.
4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.
5. Under **I/O and Communications** tab, click **Configure Protocols** > **IEC60870-5-104 Outstation**.
6. Click **Add a Master**, and the **Add IEC60870-5-104 Master** dialog appears.
7. Select **Ethernet port** and **Master Index**.

   TIP: A maximum of five IEC60870-5-104 masters can be supported per project.

8. Select **Enable Channel Redundancy**.

   NOTE: This option is ONLY available for Ethernet port ETH1.

9. Click **OK** to add a master.

   If you select **Enable Channel Redundancy**, both ports ETH1 and ETH2 appear. They share a single configuration form at ETH1.
10. In the **General** group, configure the following parameter.

Parameter	Description
TCP Port	Configure TCP port number, ranging from 1 to 65535. The default value is 2404.

11. In the **Link Layer Parameters** group, configure the following parameters.

Parameter	Description
Timeout	**t1**: Time out of send or test Application Protocol Data Units (APDUs), ranging from 1000 ms to 255000 ms.
	t2: Time out for acknowledges in case of no data messages, ranging from 1000 ms to 255000 ms.
	t3: Time out for sending test frames in case of a long idle state, ranging from 1000 ms to 255000 ms.
K/W	**K**: Maximum difference receive sequence number to send state variable, ranging from 1 to 32767.
	W: Latest acknowledge after receiving w I format APDUs, ranging from 1 to 32767.

12. In the **Application Parameters** group, configure the following paramters.

Parameter	Description
ASDU Address	Length of common address of Application-layer Service Data Unit (ASDU), ranging from 1~65535. The default value is 1.
Mapping	Select the required mapping table from the drop-down list. If the Mapping is empty, you must add a mapping table first. See Adding an IEC60870-5-104 Outstation mapping table for more information. **For redundant channel**, the same mapping table must be selected on multiple ports. For

Parameter	Description
	example, this could be used when a SCADA system communicates through 2 ports in a redundant arrangement. **For individual channel**: One mapping table can be used for multiple ports.
Cyclic Interval	The interval of cyclic data transmission via the IEC 60870-5-104 port.
Select-Before-Operate Timeout	Represents the maximum time (in seconds) allowed for the operation to be executed. If the operation is a direct command, the time is only from the time the "operation" is sent to the time the confirmation is received from the other end. If it is "Select Before Operate", it is the time when the "Select" command is sent to the completion of the execute command by the controller sending a confirmation packet.
Clock Synchronization	Enable time synchronization from the IEC60870-5-104 master. **NOTE:** Only one master can be enabled time synchronization.
Single Points Event	Used to report events related to a single point with two options: • Sequence Event: stores all the value changes during the communication loss with the master. • Current: stores the latest value during the communication loss with the master. • Supported data type : BOOL
Double Points Event	Used to report events related to a double point with two options:

Parameter	Description
	• Sequence Event: stores all the value changes during the communication loss with the master. • Current: stores the latest value during the communication loss with the master. • Supported data type : SINT , USINT
Analog Point (Scaled) Event	Used to report events related to an analog point (Scaled) with two options: • Sequence Event: stores all the value changes during the communication loss with the master. • Current: stores the latest value during the communication loss with the master. • Supported data type : INT , UINT
Analog Point (Normalized) Event	Used to report events related to an analog point (Normalized) with two options: • Sequence Event: stores all the value changes during the communication loss with the master. • Current: stores the latest value during the communication loss with the master. • Supported data type : INT , UINT
Analog Point (Short float) Event	Used to report events related to an analog point (Short float) with two options: • Sequence Event: stores all the value changes during the communication loss with the master. • Current: stores the latest value during the communication loss with the master. • Supported data type : REAL
Bit String Event	Used to report events related to a bit string with two options:

Parameter	Description
	• Sequence Event: stores all the value changes during the communication loss with the master. • Current: stores the latest value during the communication loss with the master. • Supported data type : DWORD
Step Event	Used to report events related to Step with two options: • Sequence Event: stores all the value changes during the communication loss with the master. • Current: stores the latest value during the communication loss with the master. • Supported data type : SINT ,USINT
Counter	Used to report events related to Counter with two options: • Sequence Event: stores all the value changes during the communication loss with the master. • Current: stores the latest value during the communication loss with the master. • Supported data type : DINT , UDINT

13. Select **Flash** or **SD card** from the drop-down list besides **Save Events to:**.
    - If you want to save events to SD card, you must allocate the space for the events first. See Preparing SD card for more information.
    - Up to 50,000 events can be saved to Flash per controller.
    - Up to 150,000 events can be saved to SD card per controller.
14. Click **Save**.

15. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
16. Click **Download** from the Home Page to load the configuration of MQTT to the controller.

CHAPTER

13 USER DEFINED PROTOCOL

See the following rules for using user defined protocol:

- User defined protocol can be bound on RS232 and RS485 ports. For each serial port, it allowed to connect one device via user defined protocol.
- Two function blocks are provided: COM_SEND and COM_RECV.
- Another function block CRC_16 can be used to handle CRC.
- You can make data type and use function blocks under library *PROCONS* to group or ungroup data frame.

Configuring User Defined Protocol

A new project is created and a controller is added to the project in ControlEdge Builder. See "Creating a project" and "Connecting a controller" in *ControlEdge Builder User's Guide* for more details.

To configure the User Defined Protocol:

For ControlEdge RTU:

1. From the Home Page, click **Configure Serial Ports** and select **RS232-1** , **RS232-2, RS485-1** or **RS485-2**.
2. Under **General**, select the target options in all fields.
3. Under **Protocol Binding**, select **User Defined** to bind it to the serial port.

   When you select this option, the **Delimiter Mode (Optional)** panel appears including three settings: **Read-interval Timeout (ms)**, **Max Length (Bytes)** and **End Delimiter (Hex)**. You can configure them optionally to validate if a data frame is sent completely.

   - **Read-interval Timeout (ms)**: The interval between the last data packet sent and the first keepalive probe, ranging from 0 to 10000 (ms). If the interval between the arrivals of any two bytes exceeds this Timeout, system regards it has already received a complete data frame.

     The default value is 0 which means this option is disabled.

- **Max Length (Bytes)**: The maximum number of bytes for a data frame, ranging from 0 to 1024. If the length of a received data frame exceeds the Max Length, system regards it has already received a complete data frame.

  The default value is 0 which means this option is disabled.

- **End Delimiter (Hex)**: Configured special characters in hexadecimal and based on bytes validates if a data frame is sent completely. If the received data frame has same characters with the End Delimiter, system regards it has already received a complete data frame.

  The default setting is blank which means this option is disabled.

4. Click **Save** to save the configuration, and click **Back** to return to the Home Page.
5. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
6. Click **Download** from the Home Page to load the configuration of User Defined protocol to the controller.

For ControlEdge PLC:

1. From the Home Page, under **I/O and Communications**, click **Configure Modules** > **Configure Serial Modules**.
2. Click **Add Serial Module**, the **Add Serial Module** dialog appears.
3. Select the **Type**, assign the **Rack** and **Slot** for the module.
4. Click **OK** to add the serial module.
5. Select a serial module. There are four serial ports to be configured, RS232-1, RS232-2, RS485-1 and RS485-2. Select the target port and configure appropriate values for the parameters.
6. Under **Protocol Binding**, select **User Defined** from the **Port Protocol** drop-down list.

   See the above corresponding information of ControlEdge RTU for **Delimiter Mode (Optional)**. But for ControlEdge PLC, the maximum number of **Max Length (Bytes)** bytes for a data frame, ranging from 0 to 532.
7. Click **Save** to save the configuration, and click **Back** to return to the Home Page.

8. Click **Connect** from the Home Page to connect a controller. For the user name and password, see "User Privileges" in *ControlEdge Builder User's Guide*.
9. Click **Download** from the Home Page to load the configuration of User Defined protocol to the controller.

Creating a data type for User Defined Protocol

Before you begin to program the function blocks, you should create a data type for the user defined protocol.

See the following table for reference of the frame structure:

No	Message	Size (Bytes)	Type	Remarks
0	STX	1	HEX	0X02
1	ADDR	2	HEX	Group ID, Tracker ID (0X01~0xFF)
2	CMD	4	HEX	SSF, BRO, CTG, ACK (0X01~-xFF)
3	DATA	4	HEX	Direction Mode Control, Wind Speed[option], GPS (HEX)
4	CRC16	2	HEX	0x??0x??
5	ETX	1	HEX	0x03
Sum		14		

And refer to the following picture as an example:

```
TYPE
    BYTE2:  ARRAY[1..2] OF BYTE;
    BYTE4:  ARRAY[1..4] OF BYTE;


    (* COMMAND FOR GPS POSITION SETTING *)
    CMD_GPS_SETTING:
    STRUCT
        STX:        BYTE;
        ADDRESS:    BYTE2;
        COMMAND:    BYTE4;
        DATA:       BYTE4;
        CRC:        BYTE2;
        ETX:        BYTE;
    END_STRUCT;

END TYPE
```

Configuring User Defined Protocol Function Block

Follow the instructions below to program the target device for the project in **IEC Programming Workspace**.

To configure a User Defined Protocol function block:

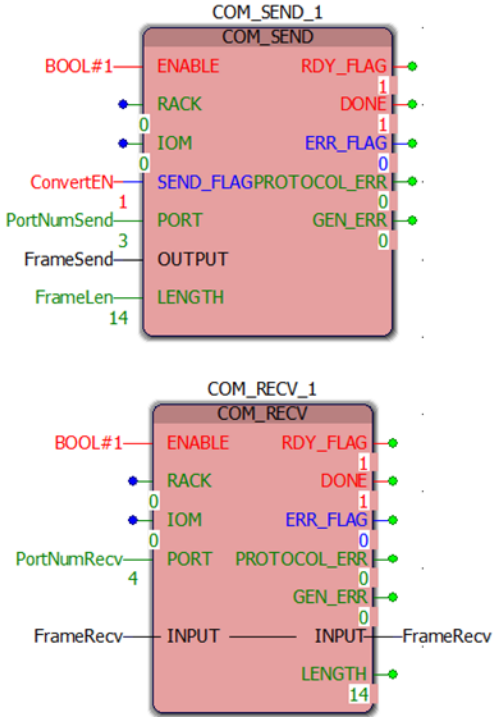
1. From the **IEC Programming Workspace**, under the **Project Tree Window**, right-click **Logical POUs** and select **Insert** > **Program**.
2. Enter the **Name** for the new POU, and select the desired programming Language. For the following steps, FBD language is used as an example.
3. Click **OK** to insert the new POU in the project tree.
4. Add a **Task** as follows:
   a. Under **Physical Hardware**, right-click **Task** and select **Insert** > **Task**.
   b. Enter the **Name** and select the task type as **CYCLIC**, and click **OK**.
   c. In the **Task settings** dialog, configure the corresponding parameters.
   d. Click **OK**.
5. Right-click the task you have inserted, and select **Insert** > **Program instance**.
6. Enter a name in the **Program instance** field.

   The program instance must not be named “RTU” or “GlobalVariable”.
7. Select the program you want to associate from the **Program type** drop-down list.

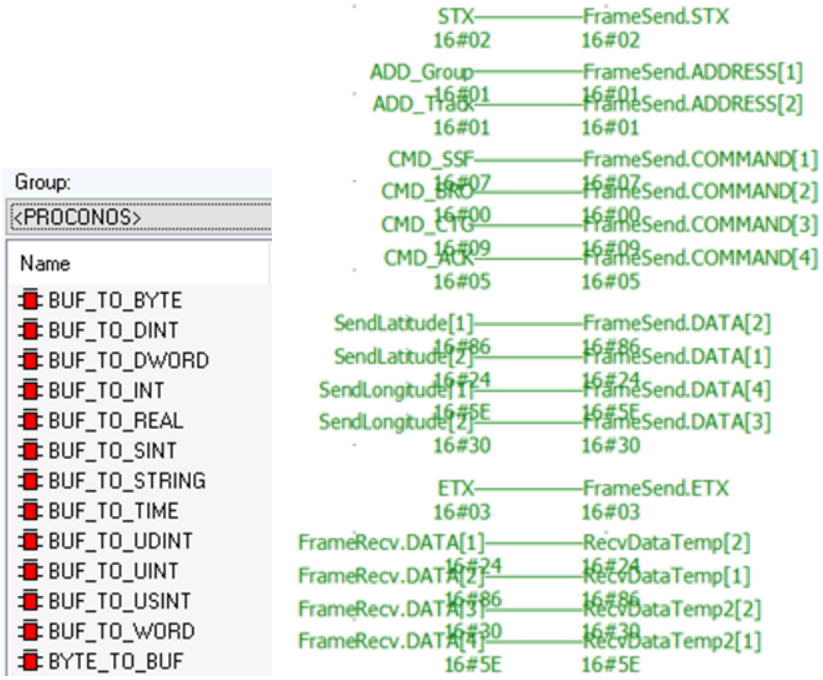
8. Right-click **Libraries** and select **Insert** > **Firmware Library**, select **UserDefined.fwl** under **UserDefined** folder. Then click **Include**.
9. Under Logical POUs, double-click the code worksheet  of the program that you have inserted.
10. From the Edit Wizard, select **UserDefined** from the **Group** list. There are two function blocks available for programming: CMD_RECV and CMD_SEND.
11. Drag the target function block into the workplace to display the function block.

    For more information about the function block, right-click it and select **Help on FB/FU** to display the embedded help.
12. Create a data type for User Defined Protocol. See Creating a data type for User Defined Protocol for more information.
13. Double-click the pin-outs of the function block to assign variables.

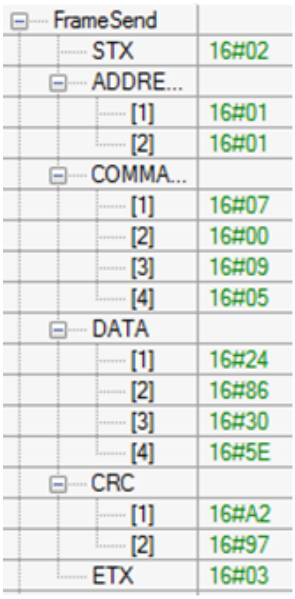
    Assign Initial value and I/O address details.

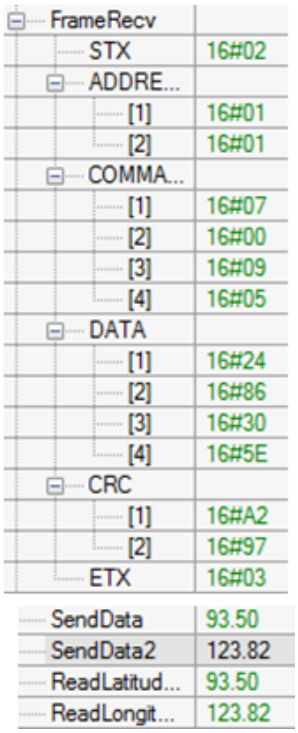


14. Use function block under *PROCONS* to group or ungroup data frame.



You can click **View-->Watch Window** and add the corresponding variables to monitor.





E. GPS Latitude/Longitude Setting

TX : 02 01 01 07 00 09 05 24 86 30 5E A2 97 03 (14 Bytes)

-, Latitude

24 86
Latitude N 93.5 North Latitude : (+) 93.50 → 9350 → 0x24 0x86

-, Longitude

30 5E
Longitude E 123.8 East Longitude (+) 123.82 → 12382 → 0x30 0x5E

15. Click **OK**.
16. Click **Make** from the toolbar to compile the programs.
17. Click **Download** from the toolbar to download the compiled programs to the controller.

NOTICES

Trademarks

Experion® is a registered trademark of Honeywell International, Inc.

ControlEdge™ is a trademark of Honeywell International, Inc.

OneWireless™ is a trademark of Honeywell International, Inc.

Other trademarks

Microsoft and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Trademarks that appear in this document are used only to the benefit of the trademark owner, with no intention of trademark infringement.

Third-party licenses

This product may contain or be derived from materials, including software, of third parties. The third party materials may be subject to licenses, notices, restrictions, and obligations imposed by the licensor. The licenses, notices, restrictions and obligations, if any, may be found in the materials accompanying the product, in the documents or files accompanying such third party materials, in a file named third_party_licenses on the media containing the product, or at http://www.honeywell.com/en-us/privacy-statement.

Documentation feedback

You can find the most up-to-date documents in the Support section of the Honeywell Process Solutions website at: https://process.honeywell.com/us/en/support/product-documents-downloads

If you have comments about Honeywell Process Solutions documentation, send your feedback to: hpsdocs@honeywell.com

Use this email address to provide feedback, or to report errors and omissions in the documentation. For immediate help with a technical problem, contact HPS Technical Support through your local Customer Contact Center, or by raising a support request on the Honeywell Process Solutions Support website.

How to report a security vulnerability

For the purpose of submission, a security vulnerability is defined as a software defect or weakness that can be exploited to reduce the operational or security capabilities of the software.

Honeywell investigates all reports of security vulnerabilities affecting Honeywell products and services.

To report a potential security vulnerability against any Honeywell product, please follow the instructions at:

https://www.honeywell.com/en-us/product-security.

Support

For support, contact your local Honeywell Process Solutions Customer Contact Center (CCC). To find your local CCC visit the website, https://process.honeywell.com/us/en/contact-us.

Training classes

Honeywell holds technical training classes that are taught by process control systems experts. For more information about these classes, contact your Honeywell representative, or see http://www.automationcollege.com.