thingworx kepware edge

Table of Contents

Table of Contents 2
Introduction 10
ThingWorx Kepware Edge System Requirements 10
API Architecture 11
ThingWorx Kepware Edge Installation 11
Application Data 12
ThingWorx Kepware Edge Licensing 13
Command Line Interface — edge_admin 15
Components and Concepts 16
What is a Channel? 16
What is a Device? 16
What is a Tag? 17
Tag Properties — General 17
System Tags 18
Property Tags 30
Statistics Tags 31
Dynamic Tags 33
Tag Properties — Scaling 34
What is a Tag Group? 35
Tag Group Properties 35
What is the Alias Map? 36
Alias Properties 36
What is the Event Log? 36
Properly Name a Channel, Device, Tag, and Tag Group 36
Getting Started 38
Managing ThingWorx Kepware Edge Services 40
Configuration Backup and Restore 40
Configuration API — Documentation Endpoint 41
Configuration API — Endpoints 41
Configuration API — Health Status Endpoint 42
Enabling Interfaces 44
Interfaces and Connectivity 45
OPC UA Interface 45
OPC UA Certificate Management 46

Configuring the ThingWorx Native Interface 46
ThingWorx Native Interface Certificate Management 48
IoT Gateway — MQTT 48
Configuring the IoT Gateway 49
Configuring Self-Signed Certificates for MQTT Agent 50
Configuration API Service 52
Security 52
Documentation 52
Configuration API Service — Concurrent Clients 52
Configuration API Service — Logging 53
Configuration API Service — Content Retrieval 54
Configuration API Service — Data 63
Configuration API Service — Invoking Services 67
Reinitialize Runtime Service 71
Configuration API Service — Project Example 71
Configuration API Service — Response Codes 73
Project Properties (via API Commands) 73
Project Properties — OPC UA 77
Channel Properties — Configuration API 79
Configuration API Service — Creating a Channel 79
Configuration API Service — Updating a Channel 80
Configuration API Service — Removing Channel 81
Device Properties — Configuration API 81
Configuration API Service — Creating a Device 82
Configuration API Service — Updating a Device 83
Configuration API Service — Removing a Device 84
Configuration API Service — Creating a Tag 84
Configuration API Service — Updating a Tag 85
Configuration API Service — Removing a Tag 86
Configuration API Service — Creating a Tag Group 86
Configuration API Service — Updating a Tag Group 87
Configuration API Service — Removing a Tag Group 88
Configuration API Service — Property Validation Error Object 88
Configuration API Service — User Management 88
Configuration API Service — Creating a User 93
Configuration API Service — Creating a User Group 93

Configuration API Service — Updating a User 94
Configuration API Service — Updating a User Group 94
Configuration API Service — Configuring User Group Project Permissions 95
Configuration API Service — Configuring Licensing Server 95
Configuration API Service — OPC UA Endpoint 97
Configuration API Service — Creating a UA Endpoint 100
Configuration API Service — Updating a UA Endpoint 100
Configuration API Service — Removing a UA Endpoint 100
Connecting with an OPC UA Client Using UaExpert 102
Event Log Messages 104
The Config API SSL certificate contains a bad signature. 104
The Config API is unable to load the SSL certificate. 104
Unable to start the Config API Service. Possible problem binding to port. 104
The Config API SSL certificate has expired. 104
The Config API SSL certificate is self-signed. 104
The configured version of TLS for the Configuration API is no longer considered secure. It is recommended that only TLS 1.2 or higher is used. 104
Configuration API started without SSL on port <port number>. 104
Configuration API started with SSL on port <port number>. 105
The <name> device driver was not found or could not be loaded. 105
Unable to load the '<name>' driver because more than one copy exists ('<name>' and '<name>'). Remove the conflicting driver and restart the application. 105
Invalid project file. 105
Unable to add channel due to driver-level failure. 106
Unable to add device due to driver-level failure. 106
Version mismatch. 106
Unable to load project <name>: 106
Unable to back up project file to '<path>' [<reason>]. The save operation has been aborted. Verify the destination file is not locked and has read/write access. To continue to save this project without a backup, deselect the backup option under Tools | Options | General and re-save the project. 107
<feature name> was not found or could not be loaded. 107
Unable to save project file <name>: 107
Device discovery has exceeded <count> maximum allowed devices. Limit the discovery range and try again. 107
<feature name> is required to load this project. 107
Unable to load the project due to a missing object. | Object = '<object>'. 108
Invalid Model encountered while trying to load the project. | Device = '<device>'. 108
Cannot add device. A duplicate device may already exist in this channel. 108

Auto-generated tag '<tag>' already exists and will not be overwritten. 108
Unable to generate a tag database for device '<device>'. The device is not responding. 108
Unable to generate a tag database for device '<device>': 109
Auto generation produced too many overwrites, stopped posting error messages. 109
Failed to add tag '<tag>' because the address is too long. The maximum address length is <number>. 109
Unable to use network adapter '<adapter>' on channel '<name>'. Using default network adapter. 109
Rejecting attempt to change model type on a referenced device '<channel device>'. 110
Validation error on '<tag>': <error>. 110
Unable to load driver DLL '<name>'. 110
Validation error on '<tag>': Invalid scaling parameters. 110
Device '<device>' has been automatically demoted. 111
Unable to load plug-in DLL '<name>'. 111
Unable to load driver DLL '<name>'. Reason: 111
Unable to load plug-in DLL '<name>'. Reason: 112
The specified network adapter is invalid on channel '%1' | Adapter = '%2'. 112
No tags were created by the tag generation request. See the event log for more information. 112
<Product> device driver loaded successfully. 112
Starting <name> device driver. 112
Stopping <name> device driver. 113
<Product> device driver unloaded from memory. 113
Simulation mode is enabled on device '<device>'. 113
Simulation mode is disabled on device '<device>'. 113
Attempting to automatically generate tags for device '<device>'. 113
Completed automatic tag generation for device '<device>'. 113
A client application has enabled auto-demotion on device '<device>'. 113
Data collection is enabled on device '<device>'. 113
Data collection is disabled on device '<device>'. 113
Object type '<name>' not allowed in project. 114
Created backup of project '<name>' to '<path>'. 114
Device '<device>' has been auto-promoted to determine if communications can be re-established. 114
Failed to load library: <name>. 114
Failed to read build manifest resource: <name>. 114
A client application has disabled auto-demotion on device '<device>'. 114
Tag generation results for device '<device>'. | Tags created = <count>. 114
Tag generation results for device '<device>'. | Tags created = <count>, Tags overwritten = <count>. 114
Tag generation results for device '<device>'. | Tags created = <count>, Tags not overwritten = <count>. 114

Access to object denied. | User = '<account>', Object = '<object path>', Permission = 115

User moved from user group. | User = '<name>', Old group = '<name>', New group = '<name>'. 115

User group has been created. | Group = '<name>'. 115

User added to user group. | User = '<name>', Group = '<name>'. 115

User group has been renamed. | Old name = '<name>', New name = '<name>'. 115

Permissions definition has changed on user group. | Group = '<name>'. 115

User has been renamed. | Old name = '<name>', New name = '<name>'. 115

User has been disabled. | User = '<name>'. 115

User group has been disabled. | Group = '<name>'. 115

User has been enabled. | User = '<name>'. 115

User group has been enabled. | Group = '<name>'. 116

Password for user has been changed. | User = '<name>'. 116

The endpoint '<url>' has been added to the UA Server. 116

The endpoint '<url>' has been removed from the UA Server. 116

The endpoint '<url>' has been disabled. 116

The endpoint '<url>' has been enabled. 116

User has been deleted. | User = '<name>'. 116

Group has been deleted. | Group = '<name>'. 116

Missing server instance certificate '<cert location>'. Please use the OPC UA Configuration Manager to reissue the certificate. 116

Failed to import server instance cert: '<cert location>'. Please use the OPC UA Configuration Manager to reissue the certificate. 116

The UA server certificate is expired. Please use the OPC UA Configuration Manager to reissue the certificate. 117

A socket error occurred listening for client connections. | Endpoint URL = '<endpoint URL>', Error = <error code>, Details = '<description>'. 117

The UA Server failed to register with the UA Discovery Server. | Endpoint URL: '<endpoint url>'. 117

Unable to start the UA server due to certificate load failure. 118

Failed to load the UA Server endpoint configuration. 118

The UA Server failed to unregister from the UA Discovery Server. | Endpoint URL: '<endpoint url>'. 118

The UA Server failed to initialize an endpoint configuration. | Endpoint Name: '<name>'. 119

The UA Server successfully registered with the UA Discovery Server. | Endpoint URL: '<endpoint url>'. 119

The UA Server successfully unregistered from the UA Discovery Server. | Endpoint URL: '<endpoint url>'. 119

Com port is in use by another application. | Port = '<port>'. 119

Unable to configure com port with specified parameters. | Port = COM<number>, OS error = <error>. 119

Driver failed to initialize. 120

Unable to create serial I/O thread. 120

Com port does not exist. | Port = '<port>'. 120
Error opening com port. | Port = '<port>', OS error = <error>. 120
Connection failed. Unable to bind to adapter. | Adapter = '<name>'. 120
Winsock shut down failed. | OS error = <error>. 121
Winsock initialization failed. | OS error = <error>. 121
Winsock V1.1 or higher must be installed to use this driver. 121
Socket error occurred binding to local port. | Error = <error>, Details = '<information>'. 121
Device is not responding. 121
Device is not responding. | ID = '<device>'. 122
Serial communications error on channel. | Error mask = <mask>. 122
Invalid array size detected writing to tag <device name>.<address>. 123
Unable to write to address on device. | Address = '<address>'. 123
Items on this page may not be changed while the driver is processing tags. 123
Specified address is not valid on device. | Invalid address = '<address>'. 123
Address '<address>' is not valid on device '<name>'. 124
This property may not be changed while the driver is processing tags. 124
Unable to write to address '<address>' on device '<name>'. 124
Socket error occurred connecting. | Error = <error>, Details = '<information>'. 124
Socket error occurred receiving data. | Error = <error>, Details = '<information>'. 125
Socket error occurred sending data. | Error = <error>, Details = '<information>'. 125
Socket error occurred checking for readability. | Error = <error>, Details = '<information>'. 125
Socket error occurred checking for writability. | Error = <error>, Details = '<information>'. 125
%s | 126
<Name> Device Driver '<name>' 126
Could not load item state data. Reason: <reason>. 126
Could not save item state data. Reason: <reason>. 126
Feature '<name>' is not licensed and cannot be used. 127
Failed to load the license interface, possibly due to a missing third-party dependency. Run in Time Limited mode only. 127
Failed to initialize licensing. Unable to load the demo file license (Error %1!x!). 127
Failed to initialize licensing. Unable to initialize the licensing identity (Error %1!x!). 127
Failed to initialize licensing. Unable to initialize trusted storage (Error %1!x!). 128
Failed to initialize licensing. Unable to initialize the licensing publisher (Error %1!x!). 128
Failed to initialize licensing. Unable to establish system time interface (Error %1!x!). 128
Failed to initialize licensing (Error <error code>) 128
Failed to process the activation response from the license server (Error: %x, Process Codes: %s, Message Codes: %s) 128
Failed to create an activation request (Error %x) 128
Request failed with license server. 128

Time Limited mode has expired. 128

Maximum device count exceeded for the lite version '<number>' license. Edit project and restart the server. 129

Maximum runtime tag count exceeded for the lite version '<number>' license. Edit client project and restart the server. 129

Type <numeric type ID> limit of <maximum count> exceeded on feature '<name>'. 130

<Object type name> limit of <maximum count> exceeded on feature '<name>'. 130

The FlexNet Licensing Service must be enabled to process licenses. Failure to enable the service results in Time Limited mode. 130

The <name> feature license has been removed. The server will enter Time Limited mode unless the license is restored before the grace period expires. 131

License for feature <name> cannot be accessed [error=<code>] and must be reactivated. 131

Feature %1 is time limited and will expire at %2. 132

Feature %1 is time limited and will expire at %2. 132

Object count limit has been exceeded on feature <name>. Time limited usage will expire at <date/time>. 132

Feature count limit exceeded on <name>. Time limited usage will expire at <date/time>. 132

Time limited usage period on feature <name> has expired. 132

Failed to obtain licenses from the license server. 132

The license for this product has expired and will soon stop functioning. Please contact your sales representative to renew the subscription. 132

Licensing for this system is currently provided by a file-based license. 132

Failed to connect to the license server. 132

Failed to return licenses to the LLS. 133

Maximum driver count exceeded for the lite version '<name>' driver license. Edit project and restart the server. 133

Connecting to the license server. 133

Successful communication with the license server. Renew interval established at %d seconds. 133

License synchronization required. Initiating request... 134

Performing initial license request to the license server. 134

Connected to license server, no changes. 134

Requesting return of all borrowed licenses... 134

Cannot add item. Requested count of <number> would exceed license limit of <maximum count>. 134

The version of component <name> (<version>) is required to match that of component <name> (<version>). 134

Maximum channel count exceeded for the lite version '<name>' driver license. Edit project and restart the server. 135

%s is now licensed. 135

Appendix — Running ThingWorx Kepware Edge in a Container 136

Appendix — Running ThingWorx Kepware Edge as an Azure IoT Edge Module 139

Index 145

Introduction

Version 1.746

ThingWorx Kepware Edge is a connectivity server that enables users to connect diverse automation devices and sensors to a wide variety of digital solutions. It offers the stability, performance, and security that is essential for industrial environments. With support for popular and secure Linux operating systems, it supports distributed architectures that improve reliability and security and reduce cost. Built by the industrial connectivity experts, ThingWorx Kepware Edge eliminates the interoperability challenges associated with implementing digital solutions.

ThingWorx Kepware Edge System Requirements

The product has been tested and verified on modern computer hardware running **Ubuntu X86_64 version 18.04 LTS**. It currently only runs on X86_64 platforms.

If running ThingWorx Kepware Edge in a container, refer to the Running in a Container for information about system requirements.

This user manual expects the user has a working knowledge of:

- Linux operating system and commands
- Command line interfaces
- Command line or API utilities, such as Postman or cURL
- ThingWorx Platform (if used)
- OPC UA configuration and connectivity (if used)
- MQTT Client interfaces and connectivity (if used)

If additional information is required, consult the vendors and websites related to those tools and technologies in use in your environment.

Prerequisites

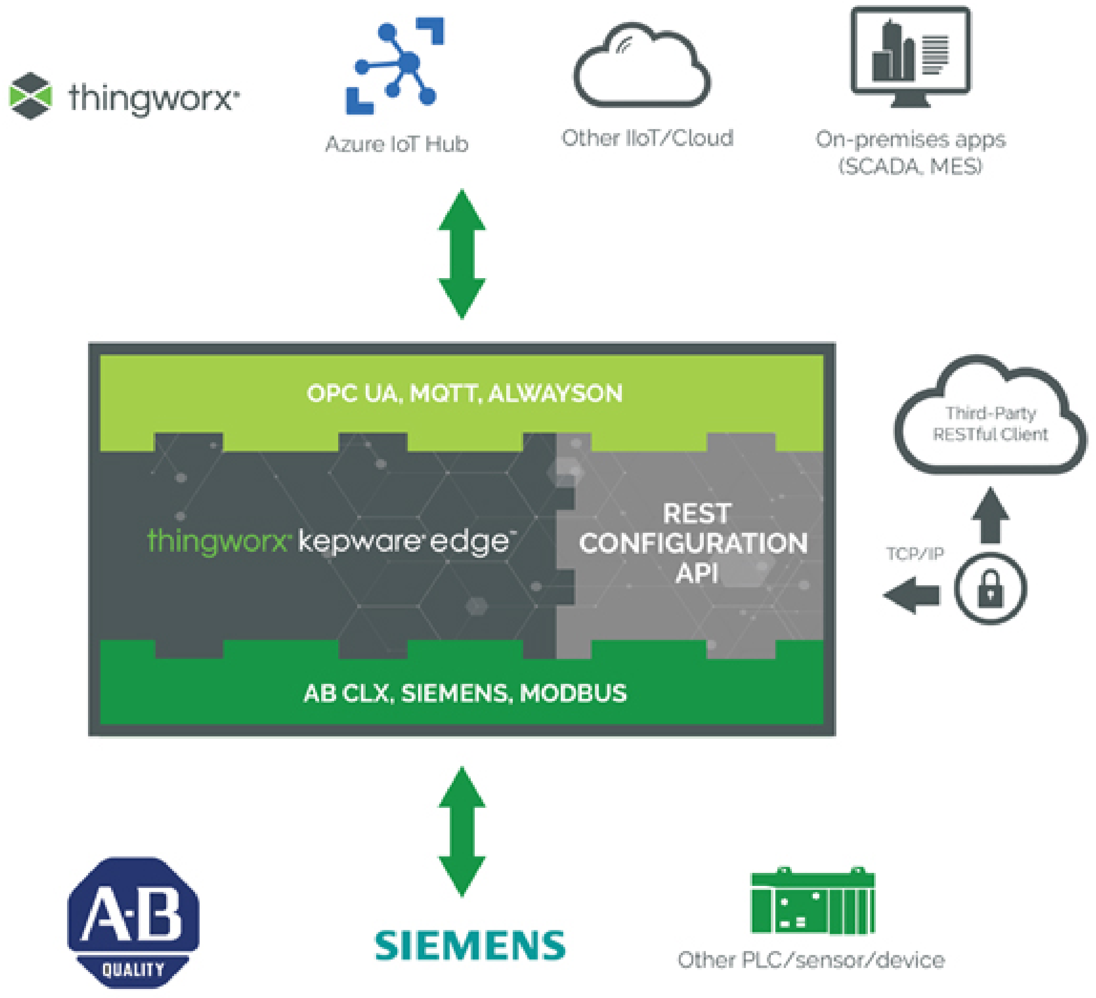
- Ubuntu 18.04 LTS
- x86-64 CPU Architecture
- Latest Linux Standard Base (LSB) package
  To install the Linux Standard Base components on Ubuntu, open a terminal and run the following command:
  ```
  $ sudo apt install lsb
  ```
- Java Runtime Environment for MQTT
  To install the Java runtime environment on Ubuntu, open a terminal and run the following command:
  ```
  $ sudo apt install default-jdk
  ```

Note: OpenJDK and Amazon Corretto have been tested and validated for running the MQTT agent.

***See Also**: The **licensing server** user manual for related system requirements.*

API Architecture

The diagram below shows the layout of the components. The Configuration API Service is installed on the same machine with the server.



ThingWorx Kepware Edge Installation

Refer to the Running in a Container *for information about installing and using* ThingWorx Kepware Edge *in a container.*

Before installing ThingWorx Kepware Edge, verify the installer hash to ensure it is the official, secure file. To generate the hash locally, run the following command and compare the results to the hash published online.

```
$ sha256sum thingworx_kepware_edge*
```

ThingWorx Kepware Edge must be installed by a user with root permissions. The installer supports both GUI and command line installations.

To install, run the following command:

```
$ ./thingworx_kepware_edge*.run
```

For all installation options, run the following command:

```
$ ./thingworx_kepware_edge*.run --help
```

Note: Ubuntu can place a lock on files needed to install software while it is checking for updates. Verify the system is updated before installing ThingWorx Kepware Edge by running the 'apt update' command.

A password should be set for the ThingWorx Kepware Edge Administrator account during installation. To skip setting a password significantly reduces the security of the installation. The Administrator account is specific to the product installation; it is not the general operating system Administrator account.

Administrator passwords must be at least 14 characters and no more than 512 characters. Passwords should be at least 14 characters and include a mix of uppercase and lowercase letters, numbers, and special characters. Choose a strong unique password that avoids well-known, easily guessed, or common passwords.

The Administrator user account password cannot be reset, but additional administrative users can be added to the Administrator user group. Best practices suggest each user with administrative access be assigned unique accounts and passwords to ensure audit integrity and continual access through role and staff changes.

Once installed, any Linux user accounts administering the ThingWorx Kepware Edge **instance must be added to the user group created during the installation, which is tkedge by default.** This allows those accounts to use the edge_admin tool and interact with the local file system to move files in and out of the secured data directory (<installation_directory>/user_data directory).

Uninstalling ThingWorx Kepware Edge

To uninstall, run the uninstall command from the <installation_directory> as root.

For a complete list of uninstall properties run the command:

```
$ sudo ./uninstall --help
```

Note: the uninstall tool leaves the <installation_directory> with the original install log and an uninstall log. This directory and these files may be removed manually.

See Also: *Command Line Interface — edge_admin*, *Application Data*

To access the command line options, run the following command:

```
$ sudo ./thingworx_kepware_edge*.run --help
```

Application Data

During installation, user_data and .config directories are created in the <installation_directory> path. The user_data directory is the relative path where all project files are saved to and loaded from using the Configuration API, as well as where files to support automatic tag generation (ATG) should be placed.

Note: All files in the user_data directory must be world readable or owned by the Linux user and group that were created during installation, which is tkedge by default.

Any authorized Linux user should be added to the user group that was created during installation to have the proper permissions to interact locally with this folder. All actions the runtime uses to interact with this folder use the Linux user configured during installation, which is tkedge by default.

Note: Any directories created in the user_data directory must be writeable by members of the ThingWorx Kepware Edge group created during installation, tkedge by default. Files in the user_data directory must be either world readable or owned by the group that was setup during installation, which is tkedge by default.

The .config directory stores currently running configuration data of the runtime, including the currently running project file, certificate information, and other instance-specific data.

Backing up the .config folder is STRONGLY RECOMMENDED as part of an application backup strategy.
*See **Configuration Backup and Restore** for more information*.

ThingWorx Kepware Edge Licensing

Licensing in ThingWorx Kepware Edge is provided on a per-tag basis across the set of supported drivers. Licensing is provided by a license server. If a license cannot be obtained from the license server, unlicensed functionality cannot be used.

See Also: ***ThingWorx Edge License Server User Manual***

The following are the available licensing levels:

- 100 Tag Limit
- 750 Tag Limit
- 1500 Tag Limit
- Unlimited

Tag count is measured across all drivers and is determined at the time of tag utilization. A tag is not considered utilized for the purpose of licensing unless there is an active client reference (a ThingWorx property binding, an OPC UA client monitored item, or an MQTT Agent Item Reference). Simulator tags and system tags are not included in the tag limit. Tags can be utilized up to the limit established by the valid license. Tags beyond this limit may be added to the server and referenced by clients, but not utilized by drivers.

Note: A server license is required for a feature to be licensed. Non-server feature licenses are provided to a client requesting them even if a server-level license is not available.

Note: Licensing for ThingWorx Kepware Edge setup through an installer or as a container requires the same process. For container implementations, any command-line functions need to be run locally within the container.

Installing a Demo License

Demo licenses are time-limited, but fully functional to allow evaluation of the software. They may be installed directly on an instance of ThingWorx Kepware Edge or distributed with the license server. Below are instructions for installing a demo license **only** on a ThingWorx Kepware Edge server.
See Also: ***ThingWorx Edge License Server User Manual***

1. Login using a local Linux user account that is a member of the ThingWorx Kepware Edge user group configured during installation, tkedge by default.
2. Use the edge_admin tool to install the demo license using the following command:
   ```
   <installation_directory>/edge_admin manage-licensing -i <file_path>
   ```
3. Restart the ThingWorx Kepware Edge runtime service using the following command to complete the licensing process:
   ```
   sudo systemctl restart tkedge_runtime.service
   ```

Configuring the License Server Connection

The license server connection can be configured using either the edge_admin command line tool or the Configuration API.

1. Set the IP address or host name of the server where the license server is running:

   Using Edge Admin:

   ```
   <installation_directory>/edge_admin manage-licensing -l <server_address>
   ```

   Using the Configuration API:

   Endpoint: (PUT)

   ```
   https://<hostname_or_ip>:<port>/config/v1/admin
   ```

   Body:

   ```
   {
   "libadminsettings.LICENSING_SERVER_NAME": "192.168.1.1"
   }
   ```

2. Import the license server certificate used when configuring the license server:

   Using Edge Admin:

   ```
   <installation_directory>/edge_admin manage-truststore -i <cert_file> licensing
   ```

3. Enable the license server connection:

   Using Edge Admin:

   ```
   <installation_directory>/edge_admin manage-licensing --lls-enable
   ```

   Using the Configuration API:

   Endpoint: (PUT)

   ```
   https://<hostname_or_ip>:<port>/config/v1/admin
   ```

   Body:

   ```
   {
   "libadminsettings.LICENSING_SERVER_ENABLE": true
   }
   ```

Note: The server can be configured to run with a self-signed certificate. This configuration is recommended for testing only.

See Also: *Configuration API Service — Configuring Licensing Server*

License Recheck

The server periodically checks the license state to verify it is up to date. The server reaches out to the license server requesting to borrow a license every specified check period when a feature in use requires a license. To trigger an immediate check of the license state, use the commands below. This feature might be helpful if new licenses have been added to the license server or if license parameters have changed.

See Also: *ThingWorx Edge License Server User Manual*

Using Edge Admin:

```
<installation_directory>/edge_admin manage-licensing --force-recheck
```

Using the Configuration API:

Endpoint: (PUT)

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ForceLicenseCheck
```

Command Line Interface — edge_admin

The edge_admin Command Line Interface (CLI) application is used to manage Configuration API settings and certificates for the server from the command line. The documentation for the edge_admin CLI may be obtained using the --help option. The following areas of functionality can be accessed through the command line.

Certificates

- Trust Store: Import / Delete / List / Trust / Reject certificates for various interfaces.
- Instance Certificate: Import / Export / Reissue instance certificates for various interfaces.
- Configuration API Settings: Enable / Disable the Configuration API and manage the ports it listens on.

Linux user accounts that interact with the edge_admin must be members of the ThingWorx Kepware Edge group that was created during installation. The edge_admin can be found at the installation location and run from the command line.

Examples

Obtain general help information and list the areas of the product that can be managed using the CLI:

```
<installation_directory>/edge_admin --help
```

View commands related to managing the configuration API:

```
<installation_directory>/edge_admin manage-cfgapi --help
```

View commands related to managing the certificates:

```
<installation_directory>/edge_admin manage-certificate --help
```

View commands related to managing the trust store:

```
<installation_directory>/edge_admin manage-truststore --help
```

To import an OPC UA certificate into the trust store:

```
<installation_directory>/edge_admin manage-truststore -i MyCertificateName.der
uaserver
```

See Also:
OPC UA Certificate Management
ThingWorx Native Interface Certificate Management

Components and Concepts

For more information on a specific server component, select a link from the list below.

What is a Channel?
What is a Device?
What is a Tag?
What is a Tag Group?
What is the Alias Map?
What is the Event Log?

What is a Channel?

A channel represents a communication medium from the PC to one or more external devices. A channel is used to represent an Ethernet-based path to target equipment.

Before adding devices to a project, users must define the channel to be used when communicating with devices. A channel and a device driver are closely tied. After creating a channel, only devices that the selected driver supports can be added to this channel.

Creating a Channel

Channels are defined by a set of properties based on the communication methods. Channels are created through the **Configuration API service**.

Channel names must be unique among all channels and devices defined in the project. *For information on reserved characters, refer to **How To... Properly Name a Channel, Device, Tag, and Tag Group**.*

Removing a Channel

To remove a channel from the project, use the **Configuration API Service**.

Displaying Channel Properties

To review the channel properties of a specific channel via the Configuration API, access the **documentation channel endpoint**.

● ***See Also***: *Channel Properties — General*

What is a Device?

Devices represent the PLCs, controllers, or other hardware with which the server communicates. The device driver that the channel is using restricts device selection.

Adding a Device

Devices are defined by a set of properties based on the protocol, make, and model. Devices are created through the **Configuration API Service**.

Device names are user-defined and should be logical for the device. This is the browser branch name used in links to access the device's assigned tags.

● *For information on reserved characters, refer to **How To... Properly Name a Channel, Device, Tag, and Tag Group**.*

Removing a Device

To remove a device from the project, use the **Configuration API Service**.

Displaying Device Properties

To review the channel properties of a specific channel via the Configuration API, access the **documentation channel endpoint**.

*For more information, refer to **Device Properties**.*

What is a Tag?

A tag represents addresses within the device with which the server communicates. The server allows both Dynamic tags and user-defined Static tags. Dynamic tags are created and stored in the client and specify device data addresses. User-defined Static tags are created and stored in the server. Static tags function as pointers to device data addresses and can be browsed from clients that support tag browsing.

*For more information, refer to **Dynamic Tags** and **Static User-Defined Tags**.*

Adding a Tag

Tags are defined by a set of properties based on the data. Tags are defined through the **Configuration API Service**.

Tag names are user-defined and should be logical for reporting and data analysis.

*For information on reserved characters, refer to **How To... Properly Name a Channel, Device, Tag, and Tag Group**.*

Removing a Tag

To remove a tag from the project; use the **Configuration API Service**.

Displaying Tag Properties

To review the tag properties of a specific channel via the Configuration API, access the **documentation channel endpoint**.

Tag Properties — General

A tag represents addresses within the device with which the server communicates. The server allows both Dynamic tags and user-defined Static tags. Dynamic tags are created and stored in the client and specify device data addresses. User-defined Static tags are created and stored in the server. Static tags function as pointers to device data addresses and can be browsed from clients that support tag browsing.

*For more information, refer to **Dynamic Tags** and **Static User-Defined Tags**.*

Name: Enter a string to represent this tag. The tag name can be up to 256 characters in length. *For information on reserved characters, refer to **How To... Properly Name a Channel, Device, Tag, and Tag Group**.*

Tip: If the application is best suited for using blocks of tags with the same names, use tag groups to separate the tags. *For more information, refer to **Tag Group Properties**.*

Description: Add context to the tag. A string of up to 255 characters can be entered for the description.

Address: Enter the target tag's driver address. The address's format is based on the driver protocol.

Data Type: Specify the format of this tag's data as it is found in the physical device. In most cases, this is also the format of the data as it returned to the client. The data type setting is an important part of how a communication driver reads and writes data to a device. For many drivers, the data type of a particular piece of data is rigidly fixed and the driver knows what format needs to be used when reading the device's data. In some cases, however, the interpretation of device data is largely in the user's hands. An example would be a device that uses 16-bit data registers. Normally this would indicate that the data is either a Short or Word. Many register-based devices also support values that span two registers. In these cases, the double register values could be a Long, DWord or 32-bit Float. When the driver being used supports this level of flexibility, users must tell it how to read data for this tag. By selecting the appropriate data type, the driver is being directed to request one or more registers.

- **Default** - Uses the driver default data type
- **Boolean** - Binary value of true or false
- **Char** - Signed 8-bit integer data
- **Byte** - Unsigned 8-bit integer data
- **Short** - Signed 16-bit integer data
- **Word** - Unsigned 16-bit integer data
- **Long** - Signed 32-bit integer data
- **DWord** - Unsigned 32-bit integer data
- **LLong** - Signed 64-bit integer data
- **QWord** - Unsigned 64-bit integer data
- **Float** - 32-bit real value IEEE-754 standard definition
- **Double** - 64-bit real value IEEE-754 standard definition
- **String** - Null-terminated Unicode string
- **BCD** - Two byte-packed BCD value range is 0-9999
- **LBCD** - Four byte-packed BCD value range is 0-99999999
- **Date** - 8-byte floating point number

Client Access: Specify whether the tag is **Read Only** or **Read / Write**. By selecting Read Only, users can prevent client applications from changing the data contained in this tag. By selecting **Read / Write**, users allow client applications to change this tag's value as needed. The **Client Access** selection also affects how the tag appears in the browse space of an OPC UA client. Many client applications allow filtering tags based on attributes. Changing the access method of this tag may change how and when the tag appears in the browse space of the client.

Scan Rate: Specify the update interval for this tag when using the **Scan Mode** option of **Respect Tag-Specified Scan Rate** within Device Properties. The server specifies an update rate on a tag per tag basis. Using the scan rate, users can tailor the bandwidth requirements of the server to suit the needs of the application. If, for example, data that changes very slowly needs to be read, there is no reason to read the value very often. Using the scan rate this tag can be forced to read at a slower rate reducing the demand on the communications channel. The valid range is 10 to 99999990 milliseconds (ms), with a 10 ms increment. The default is 100 milliseconds.

With the server's online full-time operation, these properties can be changed at any time. Changes made to tag properties take effect immediately; however, client applications that have already connected to this tag are not affected until they release and attempt to reacquire it. Utilize the User Manager to restrict access rights to server features and prevent operators from changing the properties.

System Tags

System tags provide general error feedback to client applications, allow operational control when a device is actively collecting data, and allow a channel or device's standard properties to be changed by a client application when needed.

The number of system tags available at both the channel level and device level depends on the nature of the driver being used. In addition, application-level system tags allow client applications to monitor the server's status. System tags can also be grouped according to their purpose as both status and control or property manipulation. Descriptions are as follows:

- **Status Tags** Status tags are read-only tags that provide data on server operation.
- **Parameter Control Tags**: Parameter control tags can be used to modify the server application's operational characteristics. This provides a great deal of flexibility in the client applications. By using the property control tags, users can implement redundancy by switching communications links or changing the device ID of a target device. Users can also provide access to the tags through special supervisory screens that allow a plant engineer to make changes to the communication parameters of the server if needed.

Note: If there are errors when writing to read / write system tags, verify that the authenticated user has the appropriate permissions.

The tables below include descriptions of the following:

Application-Level System Tags
Channel-Level System Tags for Ethernet Drivers
Device-Level System Tags for both Serial and Ethernet Drivers

Application-Level System Tags

Syntax Example: <*Channel Name*>.<*Device Name*>.*_System._ActiveTagCount*

Tag	Class	Description
_ActiveTagCount	Status Tag	The _ActiveTagCount tag indicates the number of tags that are currently active in the server. This is a read-only tag.
_ClientCount	Status Tag	The _ClientCount tag indicates the number of clients that are currently connected to the server. This is a read-only tag.
_Date	Status Tag	The _Date tag indicates the current date of the system that the server is running on. The format of this string is defined by the operating system date / time settings. This is a read-only tag.
_DateTime	Status Tag	The _DateTime tag indicates the GMT date and time of the system that the server is running on. The format of the string is '2004-05-21T20:39:07.000'. This is a read-only tag.

Tag	Class	Description
_DateTimeLocal	Status Tag	The _DateTimeLocal tag indicates the localized date and time of the system that the server is running on. The format of the string is '2004-05-21T16:39:07.000'. This is a read-only tag.
_Date_Day	Status Tag	The _Date_Day tag indicates the current day of the month of the system on which the server is running. This is a read-only tag.
_Date_DayOfWeek	Status Tag	The _Date_DayOfWeek tag indicates the current day of the week of the system on which the server is running. The format of the string is a number from 0 (Sunday) to 6 (Saturday). This is a read-only tag.
_Date_Month	Status Tag	The _Date_Month tag indicates the current month of the system on which the server is running. The format of the string is a number (such as "9" instead of "September"). This is a read-only tag.
_Date_Year2	Status Tag	The _Date_Year2 tag indicates the last two digits of the current year of the system on which the server is running. This is a read-only tag.
_Date_Year4	Status Tag	The _Date_Year4 tag indicates the current year of the system on which the server is running. This is a read-only tag.
_ExpiredFeatures	Status Tag	The _ExpiredFeatures tag provides a list of all server features whose time-limited usage has expired. These features are no longer operational. This is a read-only tag.
_FullProjectName	Status Tag	The _FullProjectName tag indicates the fully qualified path and file name to the currently loaded project. This is a read-only tag.
_IsDemo	Status Tag	The _IsDemo tag is no longer available as

Tag	Class	Description
		the runtime does not enter Time Limited mode in version 1.3 or higher. *See the _TimeLimitedFeatures, _LicensedFeatures, and _ExpiredFeatures tags to monitor the status of server features.*
_License_BorrowExpirationDate	Status Tag	The _License_BorrowExpirationDate tag shows the date when licenses obtained from the License Server will need to be renewed. Licenses not able to renew by this date will cease to be available on the system. This is a read-only tag.
_License_FeaturesInGrace	Status Tag	The _License_FeaturesInGrace tag shows licensed features which are past their expiration date. The licenses will soon expire permanently. This is a read-only tag.
_License_LastRequestState	Status Tag	The _License_LastRequestState tag shows the status of the last license request made to the License Server. Possible states include "Failure", "NoChanges", and "Success". This is a read-only tag.
_License_LastServerConnection	Status Tag	The _License_LastServerConnection tag shows the result of the last connection attempt the License Server. This is a Boolean tag. 1 (True) indicates a successful connection and 0 (False) indicates a failed connection. This is a read-only tag.
_LicensedFeatures	Status Tag	The _LicensedFeatures tag provides a list of all server features in use that have a valid license. If the license expires, features function through a grace period to allow users to get licensing into compliance. This is a read-only tag.
_ProductName	Status Tag	The _ProductName tag indicates the name of the underlying communication server. This is a read-only tag.

Tag	Class	Description
_ProductVersion	Status Tag	The _ProductVersion tag indicates the version of the underlying communication server. This is a read-only tag.
_ProjectName	Status Tag	The _ProjectName tag indicates the currently loaded project file name and does not include path information. This is a read-only tag.
_ProjectTitle	Status Tag	The _ProjectTitle tag is a String tag that indicates the title of the project that is currently loaded. This is a read-only tag.
_Time	Status Tag	The _Time tag indicates the current time of the system that the server is running on. The format of this string is defined by the operating system date / time settings. This is a read-only tag.
_Time_Hour	Status Tag	The _Time_Hour tag indicates the current hour of the system on which the server is running. This is a read-only tag.
_Time_Hour24	Status Tag	The _Time_Hour24 tag indicates the current hour of the system on which the server is running in a 24-hour format. This is a read-only tag.
_Time_Minute	Status Tag	The _Time_Minute tag indicates the current minute of the system on which the server is running. This is a read-only tag.
_Time_PM	Status Tag	The _Time_PM tag indicates the current AM/PM status of the system on which the server is running. This is a Boolean tag: 0 (False) indicates AM, and 1 (True) indicates PM. This is a read-only tag.
_Time_Second	Status Tag	The _Time_Second tag indicates the current second of the system on which the server is running.

Tag	Class	Description
		This is a read-only tag.
_TimeLimitedFeatures	Status Tag	The _TimeLimitedFeatures tag provides a list of all server features that are in unlicensed demo. When the time remaining expires, the feature ceases operation. This is a read-only tag.
_TotalTagCount	Status Tag	The _TotalTagCount tag indicates the total number of tags that are currently being accessed. These tags can be active or inactive. **Note**: This count does not represent the number of tags configured in the project. This is a read-only tag.

Channel-Level System Tags for Ethernet Drivers

Syntax Example: *<Channel name>._System._NetworkAdapter*

Tag	Class	Description
_AvailableNetworkAdapters	Status Tag	The _AvailableNetworkAdapters tag lists the available NICs and includes both unique NIC cards and NICs that have multiple IPs assigned to them. This tag also displays any WAN connections that are active, such as a dial-up connection. This tag is provided as a string tag and can be used to determine the network adapters available for use on this PC. The string returned contains all of the NIC names and their IP assignments. A semicolon separates each unique NIC to allow the names to be parsed within an OPC application. For a serial driver, this tag is only used if Ethernet Encapsulation is selected. This is a read-only tag.
_Description	Status Tag	The _Description tag indicates the current user-defined text description for the channel it is referencing. This is a read-only tag.
_EnableDiagnostics	Parameter Control Tag	The _EnableDiagnostics tag allows the dia-

Tag	Class	Description
		gnostic system of the driver to be enabled and disabled. The diagnostic system places a little additional burden on the driver while enabled. As such the server allows diagnostics to be enabled or disabled to improve the driver's performance. When disabled, the Diagnostics tags will not be available. *For more information, refer to **Statistics Tags**.* This is a read / write tag.
_FloatHandlingType	Parameter Control Tag	The _FloatHandlingType tag allows the current channel-level float handling to be changed. It exists in the channel-level _System folder. *For more information, refer to **Channel Properties — Advanced**.* This is a read / write tag.
_InterDeviceDelayMS	Parameter Control Tag	The _InterDeviceDelayMS tag specifies the amount of time that the channel delays sending a request to the next device after the data has been received from the current device on the same channel. The valid range is 0 to 60000 milliseconds. The default setting is 0. **Note**: This tag is only available on channels that use protocols that utilize the Inter-Device Delay. This tag is a read / write tag.
_NetworkAdapter	Parameter Control Tag	The _NetworkAdapter tag allows the current NIC adapter in use by the driver to be changed at will. As a string tag, the name of the newly desired NIC adapter must be written to this tag in string format. The string written must match the exact description to take effect. NIC names can be obtained from the ableNetworkAdapters tag listed above. For a serial driver, this tag is only used if Ethernet Encapsulation is selected. **Note**: When changing the NIC selection, the driver is forced to break all current device connections and reconnect. This is a read / write tag.

Tag	Class	Description
_WriteOptimizationDutyCycle	Parameter Control Tag	The _WriteOptimizationDutyCycle tag allows the duty cycle of the write to read ratio to be changed at will. The duty cycle controls how many writes the driver attempts for each read it performs. The _WriteOptimizationDutyCycle is defined as an unsigned long value. The valid range is 1 to 10 write per read. *For more information, refer to **Channel Properties — Write Optimizations**.* This is a read / write tag.

Device-Level System Tags for both Serial and Ethernet Drivers

Syntax Example: *<Channel Name>.<Device Name>._System._Error*

Tag	Class	Description
_AutoCreateTagDatabase	Parameter Control Tag	The _AutoCreateTagDatabase tag is a Boolean tag that is used to initiate the automatic tag database functions of this driver for the device to which this tag is attached. When this tag is set True, the communications driver attempts to automatically generate a tag database for this device. This tag does not appear for drivers that do not support Automatic Tag Database Generation. This is a read / write tag.
_AutoDemoted	Status Tag	The _AutoDemoted tag is a Boolean tag that returns the current auto-demoted state of the device. When False, the device is not demoted and is being scanned by the driver. When set True, the device is in demoted and not being scanned by the driver. This is a read-only tag.
_AutoDemotionDiscardWrites	Parameter Control Tag	The _AutoDemotionDiscardWrites tag is a Boolean tag that specifies whether or not write requests should be discarded during the demotion period. When this tag is set to False, all writes requests are performed regardless of the _AutoDemoted state. When this tag is set to True, all writes are discarded during the demotion

Tag	Class	Description
		period. This is a read / write tag.
_AutoDemotionEnabled	Parameter Control Tag	The _AutoDemotionEnabled tag is a Boolean tag that allows the device to be automatically demoted for a specific time period when the device is unresponsive. When this tag is set False, the device is never demoted. When this tag is set True, the device is demoted when the _AutoDemotedFailureCount has been reached. This is a read / write tag.
_AutoDemotedFailureCount	Parameter Control Tag	The _AutoDemotedFailureCount tag specifies how many successive failures it takes to demote a device. The _AutoDemotedFailureCount is defined as a long data type. The valid range is 1 to 30. This tag can only be written to if _AutoDemotionEnabled is set to True. This is a read / write tag.
_AutoDemotionIntervalMS	Parameter Control Tag	The _AutoDemotionIntervalMS tag specifics how long, in milliseconds, a device is demoted before re-attempting to communicate with the device. The _AutoDemotionIntervalMS is defined as a long data type. The valid range is 100 to 3600000 milliseconds. This tag can only be written to if _AutoDemotionEnabled is set to True. This is a read / write tag.
_ConnectTimeout	Parameter Control Tag	The _ConnectTimeout tag allows the timeout associated with making an IP connection to a device to be changed at will. This tag is available when either a native Ethernet driver is in use or a serial driver is in Ethernet Encapsulation mode. The _ConnectTimeout is defined as a Long data type. The valid range is 1 to 30 seconds. This is a read / write tag.
_DemandPoll	Status / Control Tag	The _DemandPoll tag issues a device read to all the active client items associated with the device. This is the equi-

Tag	Class	Description
		valent of a client performing an asynchronous device read for those items. It takes priority over any scheduled reads that are supposed to occur for items that are being actively scanned. The _DemandPoll tag becomes True (1) when written to. It returns to False (0) when the final active tag signals that the read requests have completed. Subsequent writes to the _DemandPoll tag fails until the tag value returns to False. The demand poll respects the read / write duty cycle for the channel. This is a read / write tag.
_Description	Status Tag	The _Description tag indicates the current user-defined text description for the device it is referencing. This is a read-only tag.
_DeviceId	Parameter Control Tag	The _DeviceId tag allows the ID of the device to be changed at will. The data format of the _DeviceId depends on the type of device. For most serial devices this tag is a Long data type. For Ethernet drivers the _DeviceId is formatted as a string tag, allowing the entry of an IP address. In either case, writing a new device ID to this tag causes the driver to change the target field device. This only occurs if the device ID written to this tag is correctly formatted and within the valid range for the given driver. This is a read / write tag.
_Enabled	Parameter Control Tag	The _Enabled tag provides a very flexible means of controlling the server application. In some cases, specifically in modem applications, it can be convenient to disable all devices except the device currently connected to the modem. Additionally, using the _Enabled tag to allow the application to turn a particular device off while the physical device is being serviced can eliminate harmless but unwanted communications errors in the Event Log.

Tag	Class	Description
		This is a read / write tag. **Note**: Write requests to device configuration system tags like _Enabled require editing the Project Modification permissions of the Kepware User Group associated with the client's incoming connection protocol and chosen authentication method. . OPC UA clients and other interfaces may authenticate with custom user groups and modifications should be made to those user groups as required.
_Error	Status Tag	The _Error tag is a Boolean tag that returns the current error state of the device. When False, the device is operating properly. When set True, the driver has detected an error when communicating with this device. A device enters an error state if it has completed the cycle of request timeouts and retries without a response. This is a read-only tag.
_FailedConnection	Status Tag	The _FailedConnection tag specifies that the connection failed. It is only available to specific drivers. This is a read-only tag.
_InterRequestDelay	Parameter Control Tag	The _InterRequestDelay tag allows the time interval between device transactions to be changed at will. The _InterRequestDelay is defined as a Long data type. The valid range is 0 to 30000 milliseconds. This tag only applies to drivers that support this feature. This is a read / write tag.
_RequestAttempts	Parameter Control Tag	The _RequestAttempts tag allows the number of communication attempts to be changed. The _RequestAttempts is defined as a Long value. The valid range is 1 to 10 attempts. This tag applies to all drivers equally. This is a read / write tag.

Tag	Class	Description
_RequestTimeout	Parameter Control Tag	The _RequestTimeout tag allows the timeout associated with a data request to be changed at will. The _RequestTimeout tag is defined as a Long value. The valid range is 100 to 30000 milliseconds. This tag applies to all drivers equally. This is a read / write tag.
_NoError	Status Tag	The _NoError tag is a Boolean tag that returns the current error state of the device. When True, the device is operating properly. When False, the driver has detected an error when communicating with this device. A device enters an error state if it has completed the cycle of request timeouts and retries without a response. This is a read-only tag.
_ScanMode	Status Tag	The _ScanMode tag allows clients to dictate the method used for updates. It is defined as a String value, and corresponds to the user-specified Scan Mode setting (located in device properties). "Respect client specified scan rate" has a value of "UseClientRate," "Request data no faster than *x*" has a value of "UseFloorRate," and "Request all data at *x*" has a value of "ForceAllToFloorRate." The default setting is "Respect client specified scan rate." This is a read-only tag.
_ScanRateMs	Status Tag	The _ScanRateMs tag corresponds to the _ScanMode tag, and is used when the Scan Mode is set to Request Data No Faster than Scan Rate or Request All Data at Scan Rate. This tag is defined as a DWord tag. The default setting is 1000 milliseconds. This is a read-only tag.
_SecondsInError	Status Tag	The _SecondsInError tag is a DWord tag that displays the number of seconds since the device entered an error state. This tag displays 0 when the device is not in an error state.

Tag	Class	Description
		This is a read-only tag.
_Simulated	Parameter Control Tag	The _Simulated tag is a Boolean tag that provides feedback about the simulation state of the current device. When read as True, this device is in a simulation mode. While in simulation mode, the server returns good data for this device, but does not attempt to communicate with the actual physical device. When tag is read as False, communication with the physical device is active. Changing the tag value allows clients to enable / disable simulated mode. This is a read / write tag.

The _System branch found under the DeviceName branch is always available. If referencing a system tag from a DDE application given the above example and the DDE defaults, the link would appear as "<DDE service name>| _ddedata!Channel1.Device1._System._Error".

See Also:
Property Tags
Statistics Tags

Property Tags

Property tags are used to provide read-only access to tag properties for client applications. To access a tag property, append the property name to the fully qualified tag address that has been defined in the server's tag database. For more information, refer to Tag Properties — General.

If the fully qualified tag address is "Channel1.Device1.Tag1," its description can be accessed by appending the description property as "Channel1.Device1.Tag1._Description".

Supported Property Tag Names

Tag Name	Description
_Name	The _Name property tag indicates the current name for the tag it is referencing.
_Address	The _Address property tag indicates the current address for the tag it is referencing.
_Description	The _Description property tag indicates the current description for the tag it is referencing.
_RawDataType	The _RawDataType property tag indicates the raw data type for the tag it is referencing.
_ScalingType	The _ScalingType property tag indicates the scaling type (None, Linear or Square Root) for the tag it is referencing.
_ScalingRawLow	The _ScalingRawLow property tag indicates the raw low range for the tag it is

Tag Name	Description
	referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingRawHigh	The _ScalingRawHigh property tag indicates the raw high range for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingScaledDataType	The _ScalingScaledDataType property tag indicates the scaled to data type for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingScaledLow	The _ScalingScaledLow property tag indicates the scaled low range for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingScaledHigh	The _ScalingScaledHigh property tag indicates the scaled high range for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingClampLow	The _ScalingClampLow property tag indicates whether the scaled low value should be clamped for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingClampHigh	The _ScalingClampHigh property tag indicates whether the scaled high value should be clamped for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.
_ScalingUnits	The _ScalingUnits property tag indicates the scaling units for the tag it is referencing. If scaling is set to none this value contains the default value if scaling was applied.

See Also:

Statistics Tags

System Tags

Statistics Tags

Statistics tags are used to provide feedback to client applications regarding the operation of the channel communications in the server. Statistics tags are only available when diagnostics are enabled. *For more information, refer to* Channel Diagnostics

Syntax Example: <*Channel Name*>*._Statistics._FailedReads*

Supported Statistics Tag Names

Tag Name	Description
_SuccessfulReads	The _SuccessfulReads tag contains a count of the number of reads this channel has completed successfully since the start of the application or since the last time the _Reset tag was invoked. This tag is formatted as unsigned 32-bit integer and will eventually rollover. This tag is read only.
_SuccessfulWrites	The _SuccessfulWrites tag contains a count of the number of writes this channel has completed successfully since the start of the application or since the last time the _Reset tag was invoked. This tag is formatted as an unsigned 32-bit integer and will eventually rollover. This tag is read only.

Tag Name	Description
_FailedReads	The _FailedReads tag contains a count of the number of reads this channel has failed to complete since the start of the application or since the last time the _Reset tag was invoked. This count is only incremented after the channel has failed the request based on the configured timeout and retry count for the device. This tag is formatted as an unsigned 32-bit integer and will eventually rollover. This tag is read only.
_FailedWrites	The _FailedWrites tag contains a count of the number of writes this channel has failed to complete since the start of the application or since the last time the _Reset tag was invoked. This count is only incremented after the channel has failed the request based on the configured timeout and retry count for the device. This tag is formatted as unsigned 32-bit integer and will eventually rollover. This tag is read only.
_RxBytes*	The _RxBytes tag contains a count of the number of bytes the channel has received from connected devices since the start of the application or since the last time the _Reset tag was invoked. This tag is formatted as unsigned 32-bit integer and will eventually rollover. This tag is read only.
_TxBytes	The _TxBytes tag contains a count of the number of bytes the channel has sent to connected devices since the start of the application or since the last time the _Reset tag was invoked. This tag is formatted as unsigned 32-bit integer and will eventually rollover. This tag is read only.
_Reset	The _Reset tag can be used to reset all diagnostic counters. The _Reset tag is formatted as a Boolean tag. Writing a non-zero value to the _Reset tag will cause the diagnostic counters to be reset. This tag is read / write.
_MaxPendingReads	The _MaxPendingReads tag contains a count of the maximum number of pending read requests for the channel since the start of the application (or the _Reset tag) was invoked. This tag is formatted as an unsigned 32-bit integer. The tag is read only.
_MaxPendingWrites	The _MaxPendingWrites tag contains a count of the maximum number of pending write requests for the channel since the start of the application (or the _Reset tag) was invoked. This tag is formatted as an unsigned 32-bit integer. The tag is read only.
_NextReadPriority	The _NextReadPriority is a channel-level system tag that reflects the priority level of the next read in the channel's pending read queue. Possible values are -1: No pending reads. 0: The next read is a result of a schedule-level demand poll or explicit read from a client. 1 - *n*: The next read is a result of scheduled read. This tag is read only.
_PendingReads	The _PendingReads tag contains a count of the current pending read requests for the channel. This tag is formatted as an unsigned 32-bit integer. The tag is read only.
_PendingWrites	The _PendingWrites tag contains a count of the current pending write requests for the channel. This tag is formatted as an unsigned 32-bit integer. This tag is read only.

* This statistical item is not updated in simulation mode (***See Device Properties***).

The _Statistics branch (located beneath the channel branch) only appears when diagnostics are enabled for the channel. To reference a Diagnostics tag from a DDE application, given the above example and the DDE defaults, the link would appear as: "<DDE service name>| _ddedata!Channel1._Statistics._SuccessfulReads".

See Also:

System Tags

Property Tags

Dynamic Tags

Dynamic tag addressing is a second method of defining tags that allows users to define tags only in the client application. As such, instead of creating a tag item in the client that addresses another tag item created in the server, users only need to create tag items in the client that directly accesses the device driver's addresses. On client connect, the server creates a virtual tag for that location and starts scanning for data automatically.

To specify an optional data type, append one of the following strings after the '@' symbol:

- BCD
- Boolean
- Byte
- Char
- Double
- DWord
- Float
- LBCD
- LLong
- Long
- QWord
- Short
- String
- Word

If the data type is omitted, the driver chooses a default data type based on the device and address being referenced. The default data types for all locations are documented in each individual driver's help documentation. If the data type specified is not valid for the device location, the server rejects the tag and an error posts in the Event Log.

Client Using Dynamic Addressing Example

Scan the 16-bit location "R0001" on the Simulator device. The following Dynamic tag examples assume that the project created is part of the example.

1. Start the client application and connect to the server.
2. Using the Simulator Driver, create a channel and name it Channel1. Then, make a device and name it Device1.
3. In the client application, define an item name as "Channel1.Device1.R0001@Short."

4. The client project automatically starts receiving data. The default data type for address R0001 in the Simulator device is Word. To override this, the @Short has been appended to select a data type of Short.

Note: When utilizing Dynamic tags in a client application, the use of the @[Data Type] modifier is not normally required. Clients can specify the desired data type as part of the request when registering a link for a specific data item. The data type specified by the Client is used if it is supported by the communications driver. The @[Data Type] modifier can be useful when ensuring that a communications driver interprets a piece of data exactly as needed.

Clients can also override the update rate on a per-tag basis by appending @[Update Rate].

For example, appending:
<DDE service name>| _ddedata!Device1.R0001@500 overrides just the update rate.
<DDE service name>| _ddedata!Device1.R0001@500,Short overrides both update rate and data type.

Tips:

1. The server creates a special Boolean tag for every device in a project that can be used by a client to determine whether a device is functioning properly. To use this tag, specify the item in the link as "Error." If the device is communicating properly, the tag's value is zero; otherwise, it is one.

2. If the device address is used as the item of a link such that the address matches the name of a user-defined tag in the server, the link references the address pointed to by the user-defined tag.

3. Static tags must be used to scale data in the server.

See Also:
Static Tags (User-Defined)
Designing a Project: Adding User-Defined Tags

Tag Properties — Scaling

This server supports tag Scaling, which allows raw data from the device to be scaled to an appropriate range for the application.

Type: Specify the method of scaling raw values: **Linear**, **Square Root**, or **None** to disable. The formulas for scaling types are shown below.

Type	Formula for Scaled Value
Linear	(((ScaledHigh - ScaledLow)/(RawHigh - RawLow))* (RawValue - RawLow)) + ScaledLow
Square root	(Square root ((RawValue - RawLow)/(RawHigh - RawLow))* (ScaledHigh - ScaledLow)) + ScaledLow

Raw Low: Specify the lower end of the range of data from the device. The valid range depends on the raw tag data type. For example, if the raw value is Short, the valid range of the raw value would be from -32768 to 32767.

Raw High: Specify the upper end of the range of data from the device. The Raw High value must be greater than the Raw Low value. The valid range depends on the raw tag data type.

Scaled Data Type: Specify the data type for the tag being scaled. The data type can be set to any valid OPC data type, including a raw data type, such as Short, to an engineering value with a data type of Long. The default scaled data type is Double.

Scaled Low: Specify the lower end of the range of valid resulting scaled data values. The valid range depends on the tag data type.

Scaled High: Specify the upper end of the range of valid resulting scaled data values. The valid range depends on the tag data type.

Clamp Low: Specify **Yes** to prevent resulting data from exceeding the lower end of the range specified. Specify **No** to allow data to fall outside of the established range.

Clamp High: Specify **Yes** to prevent resulting data from exceeding the upper end of the range specified. Specify **No** to allow data to fall outside of the established range.

Negate Value: Specify **Yes** to force the resulting value to be negated before being passed to the client. Specify **No** to pass the value to the client unmodified.

A client can automatically configure the range of objects (such as user input objects or displays) using the Scaling settings by accessing / changing the property tag values associated with the tag. Utilize the User Manager to restrict access rights to server features to prevent any unauthorized operator from changing these properties.

What is a Tag Group?

This server allows tag groups to be added to the project. Tag groups are used to tailor the layout of OPC data into logical groupings that fit the application's needs. Tag groups allow multiple sets of identical tags to be added under the same device: this can be convenient when a single device handles a number of similar machine segments.

Adding a Tag Group

Tag groups are defined by the set of tags contained. Tag groups are defined through the Configuration API Service.

Tag group names are user-defined and should be logical for reporting and data analysis.
For information on reserved characters, refer to How To... Properly Name a Channel, Device, Tag, and Tag Group.

Removing a Tag Group

To remove a tag from the project; use the Configuration API Service.

Displaying Tag Group Properties

To review the tag group properties of a specific tag group via the Configuration API, access the documentation channel endpoint.

Tag Group Properties

From a client standpoint, tag groups allow users to separate data into smaller tag lists, making finding specific tags easier.

Tag groups can be added at any level from the device-level down, and multiple tag groups can be nested together to fit the application's needs.

Note: With the server's online full-time operation, these properties can be changed at any time. Any changes made to the tag groups take effect immediately. If the name is changed, Clients that have already used that tag group as part of an item request are not affected until they release the item and attempt to reacquire it. New tag groups added to the project immediately allows browsing from a client. Utilize the User Manager to restrict access rights to server features to prevent operators from changing the properties.

What is the Alias Map?

The Alias Map provides both a mechanism for backwards compatibility with legacy server applications as well as a way to assign simple alias names to complex tag references. This is especially useful in client applications that limit the size of tag address paths. Although the latest version of the server automatically creates the alias map, users can add their own alias map entries to compliment those created by the server. Users can also filter the server created aliases so that the only ones visible are their own.

Alias Properties

The Alias Map allows a way to assign alias names to complex tag references that can be used in client applications.

Name: Specify the alias name, which can be up to 256 characters long. It must be unique in the alias map. *For information on reserved characters, refer to **How To... Properly Name a Channel, Device, Tag, and Tag Group**.*

Description: Enter a description of this alias to clarify data sources and reports (optional).

Mapped to: Specify the location of the alias.

Scan Rate Override: Specify an update rate to be applied to all non-OPC tags accessed using this alias map entry. The valid range is 0 to 99999990 milliseconds. The default is 0 milliseconds.
Note: When set to 0 milliseconds, the server observes the scan rate set at the individual tag level.
See Also**: **Configuration API Service — Endpoints

What is the Event Log?

The Event Log provides the date, time, and source of an error, warning, information, or security event. For more information, select a link from the list below.

Event Log Settings

Properly Name a Channel, Device, Tag, and Tag Group

When naming a channel, device, tag, or tag group, the following characters are reserved or restricted:

- Periods
- Double quotation marks
- Leading underscores
- Leading or trailing spaces

Note: Some of the restricted characters can be used in specific situations. For more information, refer to the list below.

1. Periods are used in aliases to separate the original channel name and the device name. For example, a valid name is "Channel1.Device1".
2. Underscores can be used after the first character. For example, a valid name is "Tag_1".
3. Spaces may be used within the name. For example, a valid name is "Tag 1".

Getting Started

ThingWorx Kepware Edge does not have a graphical user interface. Configuration of the server is performed using the Configuration API accessed via a REST client application / tool (not included), and the edge_admin command line interface tool. The Configuration API is used to modify all project settings and most administrative settings. The edge_admin is used to manage certificates and configure the Configuration API administrative settings.

Refer to the Running in a Container *for information about using* ThingWorx Kepware Edge *in a container.*

Additional help for the **edge_admin** tool may be found by running the tool with the '--help' option:

```
$ <installation_directory>/edge_admin --help
```

Additional help for the Configuration API may be accessed by a browser at the following **URL**:

Endpoint:

```
https://<hostname_or_ip>:<port>/config/v1/doc
```

Tip: The default port numbers are below.

Note: This version includes support for JSON-formatted documentation.

The initial API login credentials use the Administrator username and password configured during installation. For best security, a new **group** and **user** should be created via the Configuration API with only the appropriate permissions enabled.

Services:

- tkedge_configapi.service
- tkedge_eventlog.service
- tkedge_iotgateway.service
- tkedge_runtime.service

Tip: Once ThingWorx Kepware Edge is installed, verify the processes are running using the following command:

```
$ systemctl | grep tkedge
```

Ports:

- Configuration API HTTPS interface (Enabled): 57513
- Configuration API HTTP interface (Disabled by default): 57413
- OPC UA interface (Enabled by default): 49330
- Server Runtime service to IoT Gateway service (localhost only): 57213
- Server Runtime service to Configuration API service (localhost only): 32403
- Event Log service (localhost only): 56221

Service Logs

ThingWorx Kepware Edge services log information to the system journal. To view log information, run:

```
$ journalctl -u <service_name>
```

All service logs may be viewed together by running:

```
$ journalctl -u tkedge*
```

To save the log files to disk, can run the following command:

```
$ journalctl -u tkedge* >> ~/tkedgelog.txt
```

REST Configuration API Server Settings

- Endpoint: https://<hostname_or_ip>:<port>/config/
- Port: 57513 for HTTPS (57413 for HTTP)
- Authentication: Username and password of the Administrator account created during installation

A password should be set for the ThingWorx Kepware Edge Administrator account during installation. To skip setting a password significantly reduces the security of the installation. The Administrator account is specific to the product installation; it is not the general operating system Administrator account.

The Administrator user account password cannot be reset, but additional administrative users can be added to the Administrator user group. Best practices suggest each user with administrative access be assigned unique accounts and passwords to ensure audit integrity and continual access through role and staff changes.

Administrator passwords must be at least 14 characters and no more than 512 characters. Passwords should be at least 14 characters and include a mix of uppercase and lowercase letters, numbers, and special characters. Choose a strong unique password that avoids well-known, easily guessed, or common passwords.

Setting up a Project

During installation, there is an option to load a sample project. If that option was not chosen, the default project file is blank. To configure a project, use the API commands in this section to create new channels, devices, and tags. If a baseline project is helpful, the example project may be loaded after installation using these steps:

Reloading the Sample Project

1. Ensure the services are running.
2. Login using a local Linux user account that is a member of the ThingWorx Kepware Edge user group configured during installation, tkedge by default.
3. Copy the example project from <installation_directory>/examples/tke_simdemo.lpf to the <installation_directory>/user_data directory.
4. Use the configuration API to load the project using the instructions below.

Project Load Example

Load the project by performing a PUT command from a REST client to invoke request on the ProjectLoad endpoint. The name of the project file is included in the body of the request. Use basic authentication for the request. The response should include the message “Accepted” to indicate the project has been loaded.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectLoad
```

Body:

```
{
   "common.ALLTYPES_NAME": "ProjectLoad",
   "servermain.PROJECT_FILENAME":"tke_simdemo.lpf"
}
```

Authentication:

Basic Authentication with a username of administrator and the password created during installation.

Do not try to load a JSON project file generated from a server other than ThingWorx Kepware Edge as unsupported features in the project file may prevent the project from loading.

Managing ThingWorx Kepware Edge Services

ThingWorx Kepware Edge is comprised of four services:

- **Runtime**: The Runtime is the main server process. This service hosts the current project, communicates with edge devices, and provides access to data over interfaces such as OPC UA, or ThingWorx Native Interface.
- **Event Log**: The Event Log aggregates and manages log entries created by the other services.
- **Configuration API**: The Configuration API service provides a REST web service used to interact with and configure the Runtime. It also provides the ability to retrieve logs from the Event Log service.
- **IoT Gateway**: The IoT Gateway service manages MQTT agents that publish data updates from the Runtime to a MQTT broker.

Starting Each Service

The table below lists each service and the required arguments to start that service. These services all require that the working directory of the process be set to directory where the product was installed.

By default, the installer installs the services as system daemons to launch at system startup. Information below is only necessary if an alternate service invoke method is used.

Service	Command to Invoke (Shell)
Event Log	./server_eventlog
Runtime	./server_runtime
Configuration API	./config_api_service
IoT Gateway*	java -cp <installdir>/iotg/server-1.0.-jar:<installdir>/iotg/lib:<installdir>/.config/IoTGateway com.Kepware.Main -port 57213

* **Note**: The IoT Gateway must be started with port 57213. Using an alternative port number is not supported at this time.

Configuration Backup and Restore

The <installation directory>/.config directory stores currently running configuration data of the runtime, including the currently running project file, certificate information, and other instance specific data. This data can be backed up and used to restore the configuration of ThingWorx Kepware Edge if a failure occurs, such has a hardware failure to the host.

Refer to the Running in a Container for information about configuration management using ThingWorx Kepware Edge in a container.

Backing up the .config folder is STRONGLY RECOMMENDED as part of an application backup strategy.

Backing Up a Configuration

It is recommended when backing up the folder and files to maintain the ownership and access permissions that are present for the files. Errors may occur when restoring the configuration if the files and folders are not accessible by the ThingWorx Kepware Edge services. Files in this folder are owned by the Linux user and have read and write permissions for the user group configured during installation, which is “tkedge” by default for both user and group.

An example to quickly use the “cp” command to back up the .config folder:

```
sudo cp -pr /opt/tkedge/v1/.config <destination_folder>
```

Restoring a Configuration

If it is necessary to restore a configuration, the ThingWorx Kepware Edge services need to be stopped prior to copying over any files. Once ThingWorx Kepware Edge is installed, follow the steps below:

1. Stop all ThingWorx Kepware Edge services using the command:
   `sudo systemctl stop tkedge*`.
2. Delete previously stored .config folder and files.
3. Copy the backup .config folder and data to <installation_directory> (default location is /opt/tkedge/<version>).
4. Restart all ThingWorx Kepware Edge services using the commands:
   `sudo systemctl start <service name>`.

Configuration API — Documentation Endpoint

The documentation endpoint can be used to retrieve information about the various endpoints, including:

- Supported properties of the endpoint
- Child nodes of the endpoint
- Property meta data (default values, state, data ranges, etc.)
- Parameters that can be used

Note: Documentation served from the landing page is currently only available in JSON encoding.

Supported Actions

HTTP(S) Verb	Action
GET	Retrieves the current server properties

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/doc
```

Accessing the documentation endpoint URL via a browser prompts for authentication. User credentials must be used to access the documentation.

Configuration API — Endpoints

The Configuration API allows uses the following endpoint mapping scheme:

Project Connectivity Elements

```
/config/{version}/project
/config/{version}/project/aliases
/config/{version}/project/aliases/{alias_name}
/config/{version}/project/channels
```

```
/config/{version}/project/channels/{channel_name}
/config/{version}/project/channels/{channel_name}/devices
/config/{version}/project/channels/{channel_name}/devices/{device_name}
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tags
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tags/{tag_
name}
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tag_groups
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tag_groups/
{group_name}
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tag_groups/
{group_name}/tags
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tag_groups/
{group_name}/tags/{tag_name}
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tag_groups/
{group_name}/.../tag_groups
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tag_groups/
{group_name}/.../tag_groups/{group_name}/tags
/config/{version}/project/channels/{channel_name}/devices/{device_name}/tag_groups/
{group_name}/.../tag_groups/{group_name}/tags/{tag_name}
```

Plug-in Endpoints

Plug-ins are considered project extensions and are managed under the Project endpoint:

```
/config/{version}/project/{namespace}
/config/{version}/project/{namespace}/{collection}
/config/{version}/project/{namespace}/{collection}/{object_name}
```

Server Administration Endpoints

```
/config/{version}/admin
/config/{version}/admin/server_usergroups
/config/{version}/admin/server_users
/config/{version}/admin/ua_endpoints
```

Log Endpoints

```
/config/{version}/log
/config/{version}/event_log
/config/{version}/transaction_log
```

Documentation Endpoints

```
/config
/config/{version}/doc
/config/{version}/doc/drivers/{driver_name}/channels
/config/{version}/doc/drivers/{driver_name}/devices
/config/{version}/doc/drivers/{driver_name}/models
/config/{version}/doc/drivers
```

Tip: The /config/{version}/doc endpoint provides a list of all endpoints for configuration objects and the documentation endpoints for the specific object. This can be used to find definitions for all objects in the API.

Health Status Endpoint

```
/config/{version}/status
```

Configuration API — Health Status Endpoint

The health status endpoint is used to retrieve information about the Configuration API REST service status. The two values returned from a successful Health Status check are "Name" and "Healthy". Name represents the name of the server being checked and Healthy represents if the service is running or not. The

Configuration API REST Service is "healthy" if the value returned is true. If the Configuration API service is unhealthy, no response is returned.

- Supported properties of the endpoint
- Child nodes of the endpoint
- Property meta data (default values, state, data ranges, etc.)
- Parameters that can be used

Note: Documentation served from the landing page is currently only available in JSON encoding.
Documentation served from the landing page is HTML-encoded by default. To obtain JSON-encoded documentation, include an “Accept” request header with “application/json”.

Supported Actions

HTTP(S) Verb	Action
GET	Retrieves the status of the Config API REST Service

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/status
```

Accessing the status endpoint URL requires no authentication. Passing in credentials will have the same effect as its unauthenticated use.

Response Body:

```
[
   {
      “Name”: “ConfigAPI REST service”,
      “Healthy”: true
   }
]
```

Enabling Interfaces

For security reasons, only the HTTPS Configuration API endpoint and a secured OPC UA endpoint are enabled by default. The ThingWorx Native Interface and MQTT Agent are disabled by default. Interfaces are enabled or disabled using the Configuration API.

Performing a GET on the project endpoint returns a unique project ID necessary to perform a PUT successfully without using the “FORCE_UPDATE” override.

● **See Also**:
Connecting with an OPC UA Client
Configuring the ThingWorx Native Interface
Configuring the IoT Gateway

Interfaces and Connectivity

This communications server simultaneously supports the client / server technologies listed below.

Server - a software application designed to bridge the communication between a device, controller, or data source with a client application. Servers can only respond to requests made by a client.

Client - a software program that is used to contact and obtain data from a server (either on the same computer or on another computer). A client makes a request and the server fulfills the request. An example of a client would be an e-mail program connecting to a mail server or an Internet browser client connecting to a web server.

Human Machine Interface (HMI) - a software application (typically a Graphical User Interface or GUI) that presents information to the operator about the state of a process and to accept and implement the operator control instructions. It may also interpret the plant information and guide the interaction of the operator with the system.

Man Machine Interface (MMI) - a software application (typically a Graphical User Interface or GUI) that presents information to the operator about the state of a process and to accept and implement the operator control instructions. It may also interpret the plant information and guide the interaction of the operator with the system.

For more information on a specific interface, select a link from the list below.

OPC UA Interface

IoT Gateway — MQTT

OPC UA Interface

Supported Version

1.02 optimized binary TCP

Overview

OPC Unified Architecture (UA) is an open standard created by the OPC Foundation with help from dozens of member organizations. It provides an additional way to share factory floor data to business systems (from shop floor to top floor). UA also offers a secure method for remote client-to-server connectivity without depending on Microsoft DCOM. It has the ability to connect securely through firewalls and over VPN connections. This implementation of the UA server supports optimized binary TCP and the DA data model.

OPC UA Profiles

OPC UA is a multi-part specification that defines a number of services and information models referred to as features. Features are grouped into profiles, which are then used to describe the functionality supported by a UA server or client.

*For a full list and a description of each OPC UA profile, refer to **https://www.opcfoundation.org/profilereporting/index.htm**.*

Fully Supported OPC UA Profiles

- Standard UA Server Profile
- Core Server Facet
- Data Access Server Facet
- SecurityPolicy - Basic128Rsa15 (Deprecated)

- SecurityPolicy - Basic256 (Deprecated)
- SecurityPolicy - Basic256Sha256
- SecurityPolicy - None (Insecure)
- UA-TCP UA-SC UA Binary

CAUTION: Security policies Basic128Rsa15 and Basic256 have been deprecated by the OPC Foundation as of OPC UA specification version 1.04. The encryption provided by these policies is considered less secure and usage should be limited to providing backward compatibility.

Partially Supported OPC UA Profiles

- Base Server Behavior Facet

Note: This profile does not support the Security Administrator – XML Schema.

OPC UA Certificate Management

UA servers require a certificate to establish a trusted connection with each UA client. For the server to accept secure connections from a client, the client's certificate must be imported into the trusted certificate store used by the OPC UA server interface. Management of the UA certificates can be done either using the edge_admin CLI application or by saving the certificates to the configuration data folder.

Using the edge_admin CLI

To import an OPC UA certificate into the trust store:

```
./edge_admin manage-truststore -i MyCertificateName.der uaserver
```

To view the UA server trust store and the thumbprints of the certificates:

```
./edge_admin manage-truststore --list uaserver
```

Using the .config Data Folder

UA certificates can also be managed directly through .config data folder. Certificates for the UA server to use are maintained in the following directory: <installation_directory>/.config/UA/Server

Trusted certificates are located in the following directory:
<installation_directory>/.config/UA/Server/cert

Rejected certificates are located in the following directory:
<installation_directory>/.config/UA/Server/RejectedCertificates

To trust a certificate, copy the client instance certificate file into the trusted certificates directory. If a rejected certificate needs to be trusted, move the client instance certificate in the rejected certificate directory to the trusted certificates directory.

Note: The certificate files need to have read access by the installed user account, tkedge by default, for the server application to access the certificate for validation.

Configuring the ThingWorx Native Interface

To configure the ThingWorx Native Interface connection, collect the following information from the ThingWorx Platform instance to connect:

- **HOSTNAME**: Hostname or IP of machine running ThingWorx
- **PORT**: Port configured to run ThingWorx, typically port 80 for HTTP and 443 for HTTPS
- **APPKEY**: Application key configured in ThingWorx
- **THING_NAME**: Name of the Industrial Connection defined in the platform.
  Tip: If a name that does not yet exist on the platform is specified, an ephemeral thing will be created. To complete the connection, navigate to the new Thing in the platform and save.

For a list of ThingWorx interface definitions and enumerations, access the following endpoints with the REST client:

Project definitions:

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project
```

Tip: Enabling ThingWorx and configuring the connection settings can be done at the same time.

Enable ThingWorx Interface

Tip: This is already enabled if the instructions in the Quick Start Guide have been followed.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/
```

Body:

```
{
   "project_id": <project_ID_from_GET>,
   "thingworxinterface.ENABLED": true
}
```

Configure ThingWorx Test Connection Example

Note: This is a testing configuration and the use of certificates and other security measures are suggested for production systems.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project
```

Body:

```
{
   "project_id": <project_ID_from_GET>,
   "thingworxinterface.ENABLED": true,
   "thingworxinterface.HOSTNAME": "<hostname or IP>",
   "thingworxinterface.PORT": <Port Number>,
   "thingworxinterface.RESOURCE": "/ThingWorx/WS",
   "thingworxinterface.APPKEY": "<App Key>",
   "thingworxinterface.ALLOW_SELF_SIGNED_CERTIFICATE": false,
   "thingworxinterface.TRUST_ALL_CERTIFICATES": true,
   "thingworxinterface.DISABLE_ENCRYPTION": true,
   "thingworxinterface.THING_NAME": "<ThingName>"
}
```

ThingWorx Native Interface Certificate Management

ThingWorx Native Interface requires a certificate to establish a trusted connection between ThingWorx Kepware Edge and ThingWorx Platform. To create a secure connection, the ThingWorx Platform server certificate or the CA root certificate must be imported into the trusted certificate store. Management of these certificates can be accomplished using the edge_admin CLI application.

To import a the ThingWorx Platform server certificate or the CA root certificate into the trust store:

```
./edge_admin manage-truststore -i MyCertificateName.der thingworx
```

To view the ThingWorx Native Interface trust store and the thumbprints of the certificates:

```
./edge_admin manage-truststore –list thingworx
```

IoT Gateway — MQTT

Overview

The "Internet of Things" (IoT) Gateway is a built-in feature within ThingWorx Kepware Edge that allows system and device tags to be published to third-party endpoints through industry standard IP-based protocols. When the value for a configured tag changes or when a publish rate is met, an update is sent to the corresponding third-party endpoint with a configurable payload of tag ID, value, quality, and timestamp in a standard JSON format.

The IoT Gateway within ThingWorx Kepware Edge offers the following features:

- Ability to publish data consisting of a name, value, quality, and timestamp from any data source in the server (e.g. drivers, plug-ins, or system tags)
- Standard human readable JSON data format with advanced format customization options
- Support for publishing via MQTT (Message Queue Telemetry Transport) versions 3.1 and 3.1.1
- Support for MQTT subscriptions for the purpose of accepting write operations
- Configurable data collection rate, as frequent as 10 milliseconds up to once per 27.77 hours (99999990 milliseconds)
- Configurable data publish rate, as frequent as 10 milliseconds up to once per 27.77 hours (99999990 milliseconds)
- Support for authentication and TLS / SSL encryption with or without client-side certificates
- Support for user-level access based on the User Manager and Security Policies Plug-In
- Configurable payload information for integration with different third-party endpoints

Architectural Summary

The IoT Gateway is closely tied to the server’s core “tkedge_runtime.service” process, however the feature uses its own executable – “tkedge_iotgateway.service” - to manage the following functionality:

- Configuration of the MQTT client agents
- Data collection from the server runtime
- Configuration of the Gateway settings
- License enforcement
- Connection management to each third-party endpoint

- In-memory data buffering of up to 100,000 data updates buffered per agent
- Authentication and encryption management

What is MQTT?

MQTT stands for MQ Telemetry Transport. It is a publish / subscribe, extremely simple, and lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices and for mobile applications where bandwidth and battery power are at a premium *(source: www.mqtt.org)*.

See Also:
Configuring the IoT Gateway

Configuring the IoT Gateway

The IoT Gateway allows information to be conveyed to an MQTT agent. The section below describes how to configure the IoT Gateway.

IoT Gateway MQTT Agent Prerequisites

Caution: For the most secure configuration, enable ONLY those features that are being used or tested. As such, if MQTT is not being used, this section should be skipped.

1. Install a Java JRE on the machine (if this has not already been installed):
   ```
   apt install default-jdk
   ```
   Tip: OpenJDK and Amazon Corretto have been tested.
2. Once installed, verify the Java JRE version using the terminal command:
   ```
   java -version
   ```
3. Stop and restart all the ThingWorx Kepware Edge services.

MQTT Examples

Create MQTT Agent

Endpoint: (POST)

```
https://<hostname_or_ip>:<port>/config/v1/project/_iot_gateway/mqtt_clients
```

Body:

```
{
   "common.ALLTYPES_NAME": "NewMqttClient",
   "common.ALLTYPES_DESCRIPTION": "",
   "iot_gateway.AGENTTYPES_TYPE": "MQTT Client",
   "iot_gateway.AGENTTYPES_ENABLED": true
}
```

View MQTT Agents

Endpoint: (GET)

```
https://<hostname_or_ip>:<port>/config/v1/project/_iot_gateway/mqtt_clients
```

Create MQTT Agent Tag

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/_iot_gateway/mqtt_clients/NewMqttClient/iot_items
```

Body:

```
{
   "common.ALLTYPES_NAME": "Simulator_Word1",
   "iot_gateway.IOT_ITEM_SERVER_TAG": "Simulator.SimulatorDevice.Registers.Word1",
   "iot_gateway.IOT_ITEM_ENABLED": true
}
```

View MQTT Agent Tags

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/_iot_gateway/mqtt_clients/NewMqttClient/iot_items
```

Update MQTT Agent

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/_iot_gateway/mqtt_clients/NewMqttClient
```

Body:

```
{
   "project_id": <project_ID_from_GET>,
   "common.ALLTYPES_NAME": "NewMqttClient_updated",
   "common.ALLTYPES_DESCRIPTION": "Update test"
}
```

Delete MQTT Agent

Endpoint (DEL):

```
https://<hostname_or_ip>:<port>/config/v1/project/_iot_gateway/mqtt_clients/NewMqttClient_updated
```

Configuring Self-Signed Certificates for MQTT Agent

The IoT Gateway supports self-signed certificates with the MQTT agent. These agents use the Java KeyStore to manage these certificates. Use the commands below to import, list, or delete a certificate from the KeyStore.

These instructions assume the Java keytool is installed.

The default Java cacerts truststore password is “changeit”

Import certificate into the java store

```
sudo keytool -import -trustcacerts -keystore /usr/lib/jvm/<java_version>/lib/security/cacerts -alias <alias> -file <certificate>
```

List the contents of the certificate

```
keytool -list -keystore /usr/lib/jvm/<java_version>/lib/security/cacerts -alias
<alias>
```

Delete the certificate

```
sudo keytool -delete -keystore /usr/lib/jvm/<java_version>/lib/security/cacerets -
alias <alias>
```

The location of the Java Key Store used in the above commands may vary. Use the location appropriate for the local Java installation.

For more information about working with certificates using the Java keytool, consult the documentation found on the ***Oracle Java website****.*

Configuration API Service

The Configuration API allows an HTTPS RESTful client to add, edit, read, and delete objects such as channels, devices, and tags in the server. The Configuration API offers the following features:

- Object definition in standard human-readable JSON data format
- Support for triggering and monitoring actions on some objects within the server
- Security via HTTP basic authentication and HTTP over SSL (HTTPS)
- Support for user-level access based on the User Manager and Security Policies Plug-In
- Transaction logging with configurable levels of verbosity and retention

Note: This document assumes familiarity with HTTPS communication and REST concepts.

Initialization - The Configuration API is installed as a daemon and starts automatically with the system.
Operation - The Configuration API supports connections and commands between the server and REST clients.
If the Configuration API must be stopped, use the systemctl to stop the service.

Security

REST clients to the Configuration API must use HTTPS Basic Authentication. The user credentials are defined in the server User Group. Initial login to the Configuration API uses the Administrator username and the password set during installation. Additional users and groups should be created to allow the appropriate access.

The product Administrator password must be at least 14 characters and no more than 512 characters. Passwords should be at least 14 characters and include a mix of uppercase and lowercase letters, numbers, and special characters. Choose a strong unique password that avoids well-known, easily guessed, or common passwords.

The Administrator user account password cannot be reset, but additional administrative users can be added to the Administrator user group. Best practices suggest each user with administrative access be assigned unique accounts and passwords to ensure audit integrity and continual access through role and staff changes.

Individual user accounts are locked for 10 minutes after 10 successive login attempts with different, incorrect passwords.

Documentation

Please consult additional information on properties, data ranges, endpoint mapping scheme, and acceptable actions for each endpoint is available at the Configuration API Landing Page at https://<hostname_or_ip>:<port>/config/ (for default configurations).

Documentation served from the landing page is HTML-encoded by default. To obtain JSON-encoded documentation, include an “Accept” request header with “application/json”.

Configuration API Service — Concurrent Clients

The Configuration API can serve multiple REST clients at the same time. To prevent a client from editing stale configurations, the Server Runtime maintains a numeric project ID. Each time an object is edited through the Configuration API or the local Configuration client, the Project ID changes. The current project ID is returned in each GET response. PUT, POST, and DELETE requests will return a new Project ID in the response HTTPS header if the update to the project is successful. The current project ID must be specified by the client in all PUT requests.

The best practice is to issue a GET request, save the current project ID, and use that ID for the following PUT request. If only one client is used, the client may put the property “FORCE_UPDATE”: true in the PUT request body to force the Configuration API server to ignore the project ID.

Configuration API Service — Logging

Messages from the event log service can be read from a REST client by sending a GET to https://<host-name>:<port>/config/v1/event_log. The response contains comma-separated entries.

Refer to the Running in a Container *for information about additional features and using* ThingWorx Kepware Edge *in a container.*

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/event_log
```

Example Return:

```
[ {
   "timestamp": "2018-11-13T16:34:57.966",
   "event": "Security",
   "source": "ThingWorxKepwareEdge\\Runtime",    "message": "Configuration session
started by admin as Default User (R/W)." },
{
   "timestamp": "2018-11-13T16:35:08.729",
   "event": "Warning",
   "source": "Licensing",
   "message": "Feature Modbus TCP/IP Ethernet is time limited and will expire at
11/13/2019 12:00 AM."
}
…
]
```

Filtering: The Configuration API event log endpoint allows log items to be sorted or limited using filter parameters specified in the URI. The filters, which can be combined or used individually, allow the results of the log query to be restricted to a specific time period (e.g. events which occurred since a given date, events which occurred before a given date, or events that occurred between two dates). Example filtered log query:

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/event_log?limit=10&start=2016-01-
01T00:00:00.000&end=2016-01-02T20:00:00.000
```

where:

- **Limit** = Maximum number of log entries to return. The default setting is 100 entries.
- **Start** = Earliest time to be returned in YYYY-MM-DDTHH:mm:ss.sss (UTC) format.
- **End** = Latest time to be returned in YYYY-MM-DDTHH:mm:ss.sss (UTC) format.

Note: The Limit filter overrides the result of the specified time period. If there are more log entries in the time period than the Limit filter allows, only the newest specified quantity of records that match the filter criteria are displayed.

Configuration API Service — Content Retrieval

Content is retrieved from the server by issuing an HTTP(S) GET request. The URI specified in the request can target one of the following areas:

1. Online documentation (ex. https://<hostname_or_ip>:<port>/config/v1/doc or /config/v1/doc/drivers)
2. Event log entries (ex. https://<hostname_or_ip>:<port>/config/v1/event_log)
3. Transaction log entries (ex. https://<hostname_or_ip>:<port>/config/v1/transaction_log)
4. Project configuration (ex. https://<hostname_or_ip>:<port>/config/v1/project or /config/v1/project/channels/Channel1)

When targeting project configuration, a REST client can specify the type(s) of content that should be returned. In this context the word “content” refers to a category or categories of data about a collection or object instance.

By default, when a GET request is issued using an endpoint that identifies a collection, the server will return a JSON array that contains one value for each instance in the collection where each value is a JSON object that contains the properties of the instance.

By default, when a GET request is made using an endpoint that identifies an object instance, the server will return a JSON object that contains the properties of that instance.

The default behavior of these requests can be altered by specifying one or more “content” query parameters appended to the URL as in https://<hostname>:<port>/config/v1/project?content=children. The following table shows the available content types and their applicability to each endpoint type:

Content Type	Collection Endpoint	Object Instance Endpoint
properties	yes	yes
property_definitions	no	yes
property_states	no	yes
type_definition	yes	yes
children	yes	yes

The following table shows the structure of the JSON response for a given content type:

GET Request URI	JSON Response Structure

`https://<hostname_or_ip>:<port>/config/v1/project?content=properties`	`{`
	`<property name>: <value>,`
	`<property name>: <value>,`
	`...`
	`}`

`https://<hostname_or_ip>:<port>/config/v1/project?content=property_definitions`	`[`
	`{<property definition>},`
	`{<property definition>},`
	`...`
	`]`

`https://<hostname_or_ip>:<port>/config/v1/project?content=property_states`	`{`
	`“allow”:`
	`{`

GET Request URI	JSON Response Structure

	<property name>: true/false,
	<property name>: true/false,
	...
	},
	“enable”:
	{
	<property name>: true/false,
	<property name>: true/false,
	...
	}
	}

https://<hostname_or_ip>:<port>/config/v1/project?content=type_definition	{
	"name": <type name>,
	"collection": <collection name>,
	"namespace": <namespace name>,
	"can_create": true/false,
	"can_delete": true/false,
	"can_modify": true/false,
	"auto_generated": true/false,
	"requires_driver": true/false,
	"access_controlled": true/false,
	"child_collections": [<collection names>]
	}

https://<hostname_or_ip>:<port>/config/v1/project?content=children	{
	<collection name>: [
	{
	“name”: <object instance name>,
	“href”: <object instance uri>
	},
	...
	],
	<collection name>: [
	{
	“name”: <object instance name>,
	“href”: <object instance uri>
	},
	...
	],
	...
	}

Multiple content types can be specified in the same request by separating with a comma. For example, https://<hostname>:<port>/config/v1/project?content=children,type_definition. When multiple types are specified, the JSON response will contain a single object with a member for each requested content type as in:

```
{
  “properties”: <properties response structure>,
  “property_definitions”: <property definitions response structure>,
  “property_states”: <property states response structure>,
  “type_definition”: <type definition response structure>,
  “children”: <children response structure>
}
```

Type Definitions

The following table describes the members of the type definition JSON object.

Member	Type	Description
name	string	Object type name.
collection	string	Collection name. Identifies the collection in which objects of this type will exist. This name constitutes a valid endpoint that can be addressed using the REST interface.
namespace	string	Namespace that implements the object type. Objects that are implemented by the server exist in the “servermain” namespace. Other namespaces are defined by optional components such as drivers, plug-ins and client interfaces.
can_create	bool	Indicates whether or not instances of this type can be created by an end user. For example, this is false for the “Project” type because it’s not something that can be created.
can_delete	bool	Indicates whether or not instances of this type can be deleted by an end user. Again, the “Project” type is not something that can be deleted.
can_modify	bool	Indicates whether or not instances of this type can be modified by an end user. For example, the server has some auto-generated objects that exist to create a child collection only and do not themselves have any modifiable properties.
auto_generated	bool	If true, instances of this type are auto-generated by the server. Typically objects of this type will have the previous three members defined as “false”.
requires_driver	bool	True if instances of this type cannot be created without supplying the name of an installed driver.
access_controlled	bool	True if the server provides group-level access control over the CRUD operations that can be executed against an instance of this type *(see* ***User Manager****)*.
child_collections	array	An array of collection names that are supported as children under an object of this type. For example, if a type includes “devices” in “child_collections”, then object instances of that type will support one or more “Device” instance as a child.

Property Definitions

A property definition identifies the characteristics of a given property, including the type of data it supports, applicable ranges, default value, etc. The JSON structure of a property definition object is defined as follows:

Member	Type	Description
symbolic_name	string	Identifies the property by canonical name in the form <namespace>.<property name>.
display_name	localized string	The name the property would have if shown in the Server Configuration property editor. Value will be returned in the language the server is currently configured to use.

Member	Type	Description
display_ description	localized string	The description the property would have if shown in the Server Configuration property editor. Value will be returned in the language the server is currently configured to use.
read_only	Boolean	True if the property is informational, not expected to change once initially defined.
type	string	Determines the data type of the property value (see “Property Types” below).
minimum_ value	number or null (applies to numeric types)	Minimum value the property can have to be considered valid. If null, there is no minimum.
maximum_ value	number or null (applies to numeric types)	Maximum value the property can have to be considered valid. If null, there is no maximum.
minimum_ length	number (applies to strings only)	Minimum length a string value may have. 0 means no minimum.
maximum_ length	number (applies to strings only)	Maximum length a string value may have. -1 means no maximum.
hints	arrays of strings (applies to strings only)	An array of possible choices that may be assigned to the property value. This member not included if no hints exist.

enumeration	object (applies to enumerations only)	For enumeration properties, this object identifies the valid name / value pairs the enumeration can have. Structure is as follows:
		`{`
		`<name>: number,`
		`<name>: number,`
		`...`
		`}`

allow	array of objects	Defines a conditional dependency on one or more other properties that determines whether this property is relevant. Properties that are not allowed are not shown in the Server Configuration property editor (see “Allow and Enable Conditions” below).
enable	array of objects	Defines a conditional dependency on one or more other properties that determines whether this property should be enabled for the client to change. Properties that are not enabled are grayed out in the Server Config property editor (see “Allow and Enable Conditions” below).

To get specific information about the property definitions of a specific endpoint, add "?content=property_definitions" to the end of the URL of a GET request.

For example, to get the property definitions for a channel named Channel1 with the server running on the local host, the GET request would be sent to:

Endpoint:

```
https://<hostname_or_ip>:<port>/config/v1/channels/Channel1?content=property_definitions
```

The returned JSON block would look something like the following:

```
[
  {
    "symbolic_name": "common.ALLTYPES_NAME",
    "display_name": "Name",
    "display_description": "Specify the identity of this object.",
    "read_only": false,
    "type": "String",
    "default_value": null,
    "minimum_length": 1,
    "maximum_length": 256
  },
  {
    "symbolic_name": "common.ALLTYPES_DESCRIPTION",
    "display_name": "Description",
    "display_description": "Provide a brief summary of this object or its use.",
    "read_only": false,
    "type": "String",
    "default_value": null,
    "minimum_length": 0,
    "maximum_length": 255
  },
…
```

Property Types

The following table describes the different values that a property definition may contain for the “type” member. The “Value Type” identifies what JSON type the property value should have.

Type Name	Value Type	Description
AllowDeny	bool	Describes a property that contains the choices “Allow”=true and “Deny”= false.
EnableDisable	bool	Describes a property that contains the choices “Enable”=true and “Disable”= false.
YesNo	bool	Describes a property that contains the choices “Yes”=true and “No”= false.
String	string	Generic string. Properties of this type include minimum_length and maximum_length specifiers.
StringArray	array	Array of strings. Properties of this type include minimum_length and maximum_length specifiers that apply to the strings themselves, not the length of the array.
Password	string	Obfuscated string that contains a password. When changing the value of a property of this type, a plain-text password is expected. Password values should only be changed over a secure connection. ● The Administrator password must be at least 14 characters and no more than 512 characters.

Type Name	Value Type	Description
LocalFileSpec	string	A fully qualified file specification in the local file system.
UncFileSpec	string	A fully qualified file specification in a network location.
LocalPathSpec	string	A fully qualified path specification in the local file system.
UncPathSpec	string	A fully qualified path specification to a network location.
StringWithBrowser	string	Describes a property that has a string value (normally chosen from a collection of dynamically generated strings).
Integer	number	Unsigned 32-bit integer value.
Hex	number	Unsigned 32-bit integer value intended to be displayed / edited in hexadecimal notation.
Octal	number	Unsigned 32-bit integer value intended to be displayed / edited in octal notation.
SignedInteger	number	Signed 32-bit integer value.
Real4	number	Single precision floating point value.
Real8	number	Double precision floating point value.
Enumeration	number	One of the possible numeric values from the “enumeration” member of the property definition.
PropArray	object	Describes a structure containing members that each have a fixed-length array of values.
TimeOfDay	number	Integer value containing the number seconds since midnight that would define a specific time of day.
Date	number	Unix time value that specifies midnight on a given date.
DateAndTime	number	Unix time value that specifies a specific time on a given date.
Blob	array	Array of byte values that represents an opaque collection of data. Data of this type originates in the server and is hashed to prevent modification.

Allow and Enable Conditions

For definitions that contain allow and/or enable conditions, this is the structure they would have in the JSON:

```
<condition>:
[
  {
    “depends_on”: <property name>
    “operation”: “==” or “!=”
    “value”: <value>
  },
...
]
```

Each condition identifies another property that is a dependent and how it depends as equal or not equal to the value of that property. More than one dependency can exist, either on the same property or different ones. If multiple exist, the “operation” will always be the same. Evaluation of the expression to determine the state of the condition when multiple dependencies exist is a logical “or” for “==” and a logical “and” for “!=”.

When using “content=property_states”, the returned JSON describes the outcome of the evaluation of these conditions (if they exist) for each property.

Filtering

Project configuration collection requests (i.e. https://<hostname>:<port>/config/v1/project/channels) can be filtered by providing a filter query parameter on the URL. If a filter value is specified, the query returns only those objects that contain the filter value. The collection can be filtered by the Name or Description property. The request only returns those objects where the Name or Description property contains the filter value. The following example demonstrates the filter query parameter:

Filter channel list by channels that contain the text "_Siemens" through:

```
https://<hostname_or_ip>:<port>/config/v1/project/channels?filter=_Siemens
```

This only returns channel objects that include the string “_Siemens” in the name or description field.

Sorting

Project configuration collection requests (i.e. https://<hostname>:<port>/config/v1/project/channels) can be sorted by any property. To request sorting, specify a property name and the sort order (ascending or descending). The following examples demonstrate the query parameters for sorting.

Sort Channels by description, ascending:

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels?sortOrder=ascending&sortProperty&=common.ALLTYPES_
DESCRIPTION)
```

Sort Devices by tag count, descending:

```
https://<hostname_or_ip>:<-
port>/-
con-
fig/v1/-
pro-
ject/channels/Simulator/devices?sortOrder=descending&sortProperty=servermain.DEVICE_
STATIC_TAG_COUNT)
```

Tip: Sorting by a string type property value, such as common.ALLTYPES_NAME, sorts objects by number ordering (e.g. “A1”, “A10”, “A11”, “A100”). Sorting by a numeric type property value, such as servermain.CHANNEL_UNIQUE_ID, sorts objects by numeric value (e.g. 1, 2, 10, 20).

Pagination Parameters

During content retrieval (GET requests) on project configuration endpoints, collections can be paginated to break up a response into multiple pages. Pagination is enabled when supplying the pageNumber and / or pageSize parameters:

- pageNumber: Represents the page index being accessed from a paginated response. The page number must be an integer value between 1 and 2147483647. If this parameter is not specified but pageSize is, the first page of the paginated response is returned by default.
- pageSize: Represents the number of objects that are shown on a page in paginated responses. The page size must be an integer value between 1 and 2147483647. If this parameter is not specified but pageNumber is, 10 items per page are returned by default.

Below are examples of adding the pagination parameters to a Project Configuration endpoint:

- Requesting both pageSize and pageNumber:

  ```
  https://<hostname_or_ip>:<port>/config/v1/channels/?pageNumber=1&pageSize=1
  ```

- Requesting the specified number of items with only the pageSize parameter:

  ```
  https://<hostname_or_ip>:<port>/config/v1/channels/?pageSize=1
  ```

  Note: without specifying the pageNumber parameter, the first page of results is returned.
- Requesting the specified page with only the pageNumber parameter:

  ```
  https://<hostname_or_ip>:<port>/config/v1/channels/?pageNumber=2
  ```

  Note: without specifying the pageSize parameter, up to 10 items are returned for the specified page.

When information is paginated, an additional object is appended to the body of the collection being retrieved. Here is an example of pagination information returned with the body of a paginated response:

```
“pageIndex”: 1,
“totalPages”: 1,
“totalCount”: 1,
“hasPreviousPage”: false,
“hasNextPage”: false
```

Definitions for the returned pagination information:

- pageIndex: An integer representing page being accessed. This page contains a subset of content returned from an unpaginated request. The pageIndex value is the same as the pageNumber parameter.
- totalPages: The total integer number of pages used to present the collection content
- totalCount: The number of objects within the entire collection.
- hasPreviouPage: A Boolean value returning true if there are any prior pages with content before the page being accessed and false otherwise.
- hasNextPage: A Boolean value returning true if there is another page containing objects after the page being accessed and false otherwise.

The table below describes the pagination behavior based on the parameters supplied in the request:

pageNumber	pageSize	Paginated?	Page Index Returned	Items Per Page
N/A	N/A	False	N/A	Total
x	y	True	x	Up to y
x	N/A	True	x	10
N/A	y	True	1	Up to y

If no pagination parameters are specified, requests return the entire JSON response body and no pagination information. Below is an example of a non-paginated request and response:

Endpoint:

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/
```

Example JSON response where collection of object size N=2:

```
[
 {
 Object Information
   },
   {
 Object Information
```

```
 }
]
```

If the pageNumber and/or pageSize pagination parameters are specified, requests return a subset of the entire JSON response body with pagination information. Below is an example of a paginated request and response.

Endpoint:

```
https://<hostname_or_ip>:<port>/config/v1/project/channels? pageNumber=1&pageSize=1
```

Example JSON response where collection of object size N=2:

```
[
   {
   Object Information
   },
   {
   "pageIndex": 1,
   "totalPages": 2,
   "totalCount": 2,
   "hasPreviousPage": false,
   "hasNextPage": true
   }
]
```

If a collection is empty and pagination is specified, only the pagination information is returned in the JSON response body:

Endpoint:

```
https://<hostname_or_ip>:<port>/config/v1/project/channels? pageNumber=1&pageSize=1
```

Example JSON response where collection of object size N=0:

```
[
   {
   "pageIndex": 1,
   "totalPages": 0,
   "totalCount": 0,
   "hasPreviousPage": false,
   "hasNextPage": false
   }
]
```

Pagination only works for collections of objects. If the JSON payload contains a single object instance, pagination information is not appended to the response.

Endpoint:

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/<channel_name>? pageNumber=1&pageSize=1
```

Note: there is only one channel created in this instance.

Example JSON response where Just an object Instance is returned:

```
[
   {
   Object Information
   }
]
```

Configuration API Service — Data

The Configuration API Service receives requests in standard JSON format from the REST client. These requests are consumed by the server and broken down into create, read, update, or delete commands.

Please consult additional information on properties, data ranges, endpoint mapping scheme, and acceptable actions for each endpoint is available at the Configuration API Landing Page at https://<hostname_or_ip>:<port>/config/ (for default configurations).

Documentation served from the landing page is HTML-encoded by default. To obtain JSON-encoded documentation, include an "Accept" request header with "application/json".

Object names containing spaces, or other characters disallowed in URL formatting, must be percent-encoded to be correctly interpreted by the Configuration API. Percent encoding involves replacing disallowed characters with their hexadecimal representation. For example, an object named 'default object' is percent-encoded as default%20object. The following characters are not permitted in a URL and must be encoded:

space	!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%20	%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

All leading and trailing spaces are removed from object names before the server validates them. This can create a discrepancy between the object name in the server and the object name a user provides via the Configuration API. Users can send a GET on the parent object after sending a PUT/POST to verify the new or modified object name in the server matches what was sent via the API.

An attempt to perform a POST/PUT/DELETE with the API as a non-admin user fails if a user has the server configuration open at the same time. The error is a 401 status code (unauthorized). Only one user can write to the runtime at a time; the API cannot take permissions from the server configuration if it has insufficient credentials.

Create an Object

An object can be created by sending an HTTPS POST request to the Configuration API. When creating a new object, the JSON must include required properties for the object (ex. each object must have a name), but doesn't require all properties. All properties not included in the JSON are set to the default value on creation.

Example POST JSON body:

```
{
   "<Property1_Name>": <Value>,
   "<Property2_Name>": <Value>,
   "<Property3_Name>": <Value>
}
```

Create Multiple Objects

Multiple objects may be added to a given collection by including the JSON property objects in an array.

Example POST JSON body:

```
[
 {
   "<Property1_Name>": <Value>,
   "<Property2_Name>": <Value>,
   "<Property3_Name>": <Value>
 },
 {
   "<Property1_Name>": <Value>,
   "<Property2_Name>": <Value>,
   "<Property3_Name>": <Value>
 }
]
```

When a POST includes multiple objects, if one or more cannot be processed due to a parsing failure or some other non-property validation error, the HTTPS status code 207 (Multi-Status) will be returned along with a JSON object array containing the status for each object in the request.

For example, if two objects are included in the request and the second one specifies a non-validation error (in this case a parsing error), two objects are output. One is a success, and the other is an error:

```
[
 {
   “code”: 201,
   “message”: “Created”
 },
 {
   “code”: 400,
   “message”: "Failed to parse JSON document at line 21: Property servermain.CHANNEL_
WRITE_OPTIMIZATIONS_DUTY_CYCLE cannot be converted to the expected type."
 }
]
```

If the error is a property validation error, the same HTTPS status code 207 is returned, but two error objects are returned rather than one per property validation error. The basic error object contains the error code and error message (such as above). The more comprehensive error message returns the property that caused the error, the error description, the line of input that caused the error, the error code, and error message.

Tip: When there is a property validation error on multi-object requests, the order of the objects returned maintains the sequential order of the input.

For example, if two objects are included in the request and the second one specifies the same name as the first, this is a property validation error:

```
{
   “property”: “common.ALLTYPES_NAME”,
   “description”: “The name “Channel1” is already used.”,
   “error_line”: 7,
   “code”: 400,
   “message”: "Validation failed on property common.ALLTYPES_NAME in object defin-
ition at line 7: The name 'Channel1' is already used."
}
```

The first object returned is a response to successful creation of Channel1, while the second and third response objects correspond to the property validation error.

Create an Object with Child Hierarchy

An object may be created with a full child object hierarchy beneath it. To do this, include that hierarchy in the POST request just as it would appear when saved in a JSON project file.

For example, to create a channel with a device underneath it, the following JSON could be used:

```
{
 "common.ALLTYPES_NAME": "Channel1",
 "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
 "devices":
 [
 {
 "common.ALLTYPES_NAME": "Device1",
 "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
 "servermain.DEVICE_MODEL": 0
 }
 ]
}
```

There is no response body when a child object is created unless there is an error during creation (such as a parsing error or property validation error). A response header with the Project_ID is returned with a successful request. That response header includes the Project_ID value, which is a new Project_ID after successful object creation.

Header Information

Key	Value
Connection	keep-alive
Content-Length	0
Project_ID	12345678

Read an Object

An object can be read by sending an HTTPS GET request to the Configuration API. All object properties are returned on every GET request and each object includes a Project_ID. The Project_ID property is used to track changes in the configuration and is updated on any change from the Configuration API or a server configuration client. This property should be saved and used in all PUT requests to prevent stale data manipulations.

Example response body:

```
{
   "<Property1_Name>": <Value>,
   "<Property2_Name>": <Value>,
   "PROJECT_ID": 12345678
}
```

The header of a successful GET request contains the Project_ID.

Header Information

Key	Value
Connection	keep-alive
Content-Length	0
Project_ID	12345678

See Also: Content Retrieval

Edit an Object

An object can be edited by sending an HTTPS PUT request to the Configuration API. PUT requests require the Project_ID or Force_Update property in the JSON body. Setting Force_Update to True ignores Project_ID validation.

Example PUT body:

```
{
   "<Property1_Name>": <Value>,
   "<Property2_Name>": <Value>,
   "PROJECT_ID": 12345678,
   "FORCE_UPDATE": true
}
```

Normally, when a PUT request succeeds and all properties are assigned successfully, there is no response body returned to the client; there is only a 200 status code to indicate success. There can be cases where a property is included in a PUT request that is not assigned to the object instance by the Server Runtime. In these cases, a response body will be generated as follows:

The header of a successful PUT request contains the new Project_ID that changed.

Header Information

Key	Value
Connection	keep-alive
Content-Length	0
Project_ID	12345678

Body:

```
{,
   "not_applied":,
   {,
   "servermain.CHANNEL_UNIQUE_ID": 2466304381
   },
   "code": 200,
   "message": "Not all properties were applied. This could be due to active client
reference or property is disallowed/disabled/read-only."
}
```

The response indicates which property or properties were not applied to the object instance where each contains the value that is actually in use. There are several possible reasons why the property value could not be applied, such as:

- The property is read-only and cannot be changed.
- There is a client reference on the object that restricts what properties can be updated.
- The property is not allowed based on the values of other properties on which this condition depends.
- The property is not enabled based on the values of other properties on which this condition depends.
- The value was transformed in some way (ex. rounded or truncated).

Delete an Object

An object can be deleted by sending an HTTPS DELETE request to the Configuration API. The Configuration API does not allow deleting multiple items on the same level with a single request (such as deleting all of the devices in a channel), but can delete an entire tree (such as deleting a device deletes all its child tags).

The header of a successful DELETE request contains the new Project_ID that changed.

Header Information

Key	Value
Connection	keep-alive
Content-Length	0
Project_ID	12345678

Errors

All Configuration API Service requests return errors in JSON format.

Example:

```
{
   "code": 400,
   "message": "Invalid property: 'NAME'."
}
```

See Also: Troubleshooting

Configuration API Service — Invoking Services

Objects may provide services if there are actions that can be invoked on the object beyond the standard CRUD (Create, Retrieve, Update, Delete) operations. Services provide an asynchronous programmatic interface through which remote clients can trigger and monitor these actions. Services can be found in a collection called 'services' underneath the object on which they operate. For example, the project load service is located at the https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectLoad endpoint as it operates on the project. Any object may provide services, so query if the service collection exists, then query the collection to see the available services.

Service Architecture

Services are designed to provide stateless interaction with the object on which they operate. Services are comprised of two components: a service and a job. The job executes the work asynchronously and provides a mechanism through which a client can monitor the job for completion or for any errors that occurred during its operation. After a job completes, it is scheduled for deletion automatically by the server; no action is required by the client to clean up the job after it completes.

Service

The service is the interface through which an action is invoked. The service exposes all parameters that can be specified during its invocation as properties. To see the available parameters, perform a HTTPS GET on the service endpoint. All properties, besides the name and description of the service, are the parameters that can be included when invoking a service. Depending on the service, some or all parameters may be required.

Invocation of a service is accomplished by performing a HTTPS PUT request on the service endpoint with any parameters specified in the body of the request. Services may limit the total number of concurrent invocations. If the maximum number of concurrent invocations has been reached, the request is rejected with an "HTTPS 429 Too Many Requests" response. If the limit has not been reached, the server responds with an "HTTPS 202 Accepted" response and the body of the response including a link to the newly created job.

Successful PUT response example:

```
{
  "code": 202,
  "message": "Accepted",
  "href": "/config/v1/project/services/ProjectLoad/jobs/job1"
}
```

Busy PUT response example:

```
{
  "code": 429,
  "message": "The server is busy. Retry the operation at a later time."
}
```

Job

The job represents a specific request accepted by the server. To check the status of a job, perform a HTTPS GET request on the job endpoint. The **servermain.JOB_COMPLETE** property represents the current state of the job as a Boolean. The value of this property remains false until the job has finished executing. If the job fails to execute for any reason, it provides the client with an appropriate error message in the **servermain.JOB_STATUS_MSG** property.

Job Cleanup

Jobs are automatically deleted by the server after a configurable amount of time. By default, after a job has completed, the client has 30 seconds to interact with it before the job is deleted. If a longer amount of time is required by the client or the client is operating over a slow connection, the client can use the **servermain.JOB_TIME_TO_LIVE_SECOND** parameter when invoking the service to increase the time-to-live up to a maximum of five minutes. Each job has its own time-to-live and it may not be changed after a job has been created. Clients are not allowed to manually delete jobs from the server, so it is best to choose the shortest time-to-live without compromising the client's ability to get the information from the job before it is deleted.

Service Automatic Tag Generation

The Automatic Tag Generation service operates under a device endpoint for a driver that supports Automatic Tag Generation. The properties that support Automatic Tag Generation for the device must be configured prior to initiating Automatic Tag Generation. *See the driver specific documentation for related properties.*

To initiate Automatic Tag Generation, a PUT is sent to the TagGeneration endpoint with a defined empty payload. In the following example, Automatic Tag Generation is initiated on Channel1/Device1.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices/Device1/services/TagGeneration
```

The response should look something like the following.

Body:

```
{
  "code": 202,
  "message": "Accepted",
  "href": "/config/v1/project/channels/Channel1/devices/Device1/services/TagGeneration/jobs/job1"
}
```

This means the request was accepted and the job was created as job1. The status of the job can be seen by querying the job. This is done by sending a GET to the job's endpoint. The GET request should look like the following.

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/-
config/v1/project/channels/Channel1/devices/Device1/services/TagGeneration/jobs/job1
```

Jobs are automatically cleaned up after their wait time has expired. This wait time is configurable.

*See the **Job Cleanup** section for more information.*

Note: Not all drivers support Automatic Tag Generation.

Tip: Automatic Tag Generation files must be located in the <installation_directory>/user_data directory. All files in the user_data directory must be world readable or owned by the ThingWorx Kepware Edge user and group that were created during installation, by default this is tkedge.

Service Project Load

Projects can be loaded by interacting with the ProjectLoad service on the ProjectLoad endpoint. First a GET request must be sent to get the Project ID to later be used in the PUT request.

The GET request should look like the following.

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectLoad
```

The server should respond with something similar to the following.

Body:

```
{
  "PROJECT_ID": 3531905431,
  "common.ALLTYPES_NAME": "ProjectLoad",
  "servermain.JOB_TIME_TO_LIVE_SECONDS": 30,
  "servermain.PROJECT_FILENAME": "",
  "servermain.PROJECT_PASSWORD": ""
}
```

To initiate the project load, a PUT request is sent to the server with the absolute path to the project file, the project file password, and the Project ID. If there is no password on the project, that field is not required. Project loading supports SLPF, LPF, and JSON file types. The request should look similar to the following.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectLoad
```

Body:

```
{
  "PROJECT_ID": 3531905431,
  "servermain.PROJECT_FILENAME": "/Absolute/Path/To/MyProject.json",
  "servermain.PROJECT_PASSWORD": ""
}
```

The server should respond with something similar to the following.

Body:

```
{
  "code": 202,
  "message": "Accepted",
  "href": "/config/v1/project/services/ProjectLoad/jobs/job1"
}
```

This means the request was accepted and the job was created as job1. The status of the job can be seen by querying the job. This is done by sending a GET to the job's endpoint. The GET request should look like the following.

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectLoad/jobs/job1
```

Jobs are automatically cleaned up after their wait time has expired. This wait time is configurable.

*See the **Job Cleanup** section for more information.*

Service Project Save

Projects can be loaded by interacting with the ProjectSave service on the ProjectSave endpoint. A GET request must be sent to get the Project ID to later be used in the PUT request. The GET request should look similar to the following.

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectSave
```

The server should respond with something similar to the following.

Body:

```
{
  "PROJECT_ID": 2401921849,
  "common.ALLTYPES_NAME": "ProjectSave",
  "servermain.JOB_TIME_TO_LIVE_SECONDS": 30,
  "servermain.PROJECT_FILENAME": ""
}
```

To initiate the project save, a PUT request is sent with the project file path and name of the file with the extension (SLPF, LPF, or JSON), the password to encrypt it with, and the Project ID. The password property is required for SLPF file and ignored otherwise. The path is relative to the Application Data Folder. The PUT request should look similar to the following.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectSave
```

Body:

```
{
  "PROJECT_ID": 2401921849,
  "servermain.PROJECT_FILENAME": "Projects/MyProject.SLPF",   "servermain.PROJECT_
PASSWORD": "MyPassword" }
```

The server should respond with something similar to the following.

Body:

```
{
  "code": 202,
  "message": "Accepted",
  "href": "/config/v1/project/services/ProjectSave/jobs/job1"
}
```

This means the request was accepted and the job was created as job1. The status of the job can be seen by querying the job. This is done by sending a GET to the job's endpoint. The GET request should look like the following.

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectSave/jobs/job1
```

Jobs are automatically cleaned up after their wait time has expired. This wait time is configurable.
*See the **Job Cleanup** section for more information.*

Reinitialize Runtime Service

The Runtime Service can be reinitialized by interacting with the ReinitializeRuntime service. To initiate the reinitialization, a PUT request is sent to the endpoint with a body that defines the service name and the job's desired Time to Live (timeout).

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ReinitializeRuntime
```

Body:

```
{ "common.ALLTYPES_NAME" = "ReinitializeRuntime", "servermain.JOB_TIME_TO_LIVE_
SECONDS" = 30 }
```

The server should respond with something similar to the following.

Body:

```
{ "code": 202, "message": "Accepted", "href": "/con-
fig/v1/project/services/ReinitializeRuntime/jobs/job1"
}
```

This means the request was accepted and the job was created as job1. The status of the job can be seen by querying the job by sending a GET to the job's endpoint. The GET request should look like the following.

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/services/ReinitializeRuntime/jobs/job1
```

Jobs are automatically cleaned up after the wait time expires. This wait time is configurable.

See Also: Job Cleanup

Configuration API Service — Project Example

Project files control the communications and data collection of the server and all connected devices. Channel and device properties are defined and saved in the project file and how they are configured can impact performance *(see Optimization)*. Tag and tag group settings saved in the project can impact how the data is available in control and monitoring displays and reports. There must always be one active open project.

Project saving and loading is restricted to the <installation_directory>/user_data directory. A local user must be a member of the ThingWorx Kepware Edge user group created during installation, tkedge by default, to be able to place files in this directory. The <installation_directory>/user_data directory is also used for loading of automatic tag generation (ATG) files.

Note: All files in the user_data directory must be world readable or owned by the ThingWorx Kepware Edge user and group that were created during installation, by default this is tkedge.

See Also: *Application Data*

Save a Project

Use a “PUT” command from a REST client to invoke the ProjectSave service and provide a unique file name for the new file. All files are loaded from and saved to the <installation_directory>/user_data directory.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/services/ProjectSave
```

Body:

```
{
   "common.ALLTYPES_NAME": "ProjectSave",
   "servermain.PROJECT_FILENAME":"myProject.json"
}
```

Note: The project is saved to: <installation_directory>/user_data/. A path may be included in the file name, such as ‘projects/MyProject.json’. Any directory that does not exist within the <installation_directory>/user_data/ directory will be created upon successfully saving a project file.

Update a Project

The typical work flow for editing a project is to read the properties using a GET, modify the properties, then write them into the body of the message using a PUT.

Read Available Device Properties Example

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/<channel_name>/devices
```

Return:

```
[
 {
   "PROJECT_ID": <project_ID_from_GET>,
   "common.ALLTYPES_NAME": <device_name>,
   "common.ALLTYPES_DESCRIPTION": "",
   "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "<driver>",
   "servermain.DEVICE_MODEL": 0,
   "servermain.DEVICE_UNIQUE_ID": <ID>,
   "servermain.DEVICE_CHANNEL_ASSIGNMENT": "<channel_name>",
   "servermain.DEVICE_ID_FORMAT": 0,
   "servermain.DEVICE_ID_STRING": "<nnn.nnn.n.n>.0",
…
 }
]
```

where nnn.nnn.n.n is the Device ID address.

Update Specific Device Properties Example

Only the properties you wish to change are needed for this step.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/<channel_name>/devices/<device_name>
```

Body:

```
{
   "project_id": <project_ID_from_GET>,
   "servermain.DEVICE_ID_STRING": "<nnn.nnn.n.n>.0"
}
```

where nnn.nnn.n.n is the Device ID address.

Configuration API Service — Response Codes

One of the following response codes may be returned from a REST request. Where possible, the body of the response contains specific error messages to help identify the cause of the error and possible solutions:

- HTTPS/1.1 200 OK
- HTTPS/1.1 201 Created
- HTTPS/1.1 202 Accepted
- HTTPS/1.1 207 Multi-Status
- HTTPS/1.1 400 Bad Request
- HTTPS/1.1 401 Unauthorized
- HTTPS/1.1 403 Forbidden
- HTTPS/1.1 404 Not Found
- HTTPS/1.1 429 Too Many Requests
- HTTPS/1.1 500 Internal Server Error
- HTTPS/1.1 503 Server Runtime Unavailable
- HTTPS/1.1 504 Gateway Timeout
- HTTPS/1.1 520 Unknown Error

*Consult the **Configuration API Service Event Log Messages***

Project Properties (via API Commands)

The project endpoint is used to manage the project running in the server. All objects within the project can be found underneath the project endpoint. *To browse the child endpoints, see **Content Retrieval**.*

The project endpoint provides a single point of access for configuring both global project settings as well as client interfaces.

*See **Client Interfaces** for detailed information on the available client interfaces and their associated settings.*

*See **Project Properties** for detailed information on the available Project Property settings.*

Supported Actions

HTTPS Verb	Action
GET	Retrieves the current project properties
PUT	Updates the project properties

Child Endpoints

Endpoint	Description
/config/v1/project/channels	Endpoint used to manage the channels in the project
/config/v1/project/_iot_gateway	Endpoint used to manage the IOT Gateway client interface configuration

Endpoint	Description
/config/v1/project/aliases	Endpoint used to manage the object aliases in the project
/config/v1/project/client_interfaces	Endpoint used to manage the various client interfaces
/config/v1/project/services	Endpoint used to access the services available to the project (*see Project Load and Project Save*)

GET /config/v1/project

Returns the set of project properties as they are configured when the request is processed.

Note: You cannot delete the project or create a new one. However, you can load a new project or save the project using the Project Load and Project Save services.

***See Also**: Project Load and Project Save)*

Resource Information

Type	Description
Resource URL	https://<hostname/port>:<port>/config/v1/project
Response Format	JSON

Parameters

Content	Returns
content=properties	Returns the project properties
content=property_definitions	Returns a detailed description for each property in the project endpoint
content=property_states	Returns the property states
content=type_definition	Returns the type definitions
content=children	Returns a collection of child endpoints underneath the project endpoint.

Properties

Property Name	Type	Description
common.ALLTYPES_DESCRIPTION	String	Provide a brief summary of this object or its use.
servermain.PROJECT_TITLE	String	Title of the project for informational purposes.
servermain.PROJECT_TAGS_DEFINED	String	Count of tags identified in the project
uaserverinterface.PROJECT_OPC_UA_ENABLE	YesNo	Enable the OPC UA server interface to accept client connections. Changes in this property require runtime reinitialization to take effect.
uaserverinterface.PROJECT_OPC_UA_DIAGNOSTICS	YesNo	Enable sending diagnostic information to the event log. **Warning**: Enabling UA diagnostics allows server users to view encrypted OPC UA client / server traffic.

Property Name	Type	Description
uaserverinterface.PROJECT_OPC_UA_ANONYMOUS_LOGIN	YesNo	Important: You must use Server Administration to define users if anonymous login is not allowed.
uaserverinterface.PROJECT_OPC_UA_MAX_CONNECTIONS	Integer	The number of simultaneous OPC UA client connections allowed by the server. Changes in this property require runtime reinitialization to take effect.
uaserverinterface.PROJECT_OPC_UA_MIN_SESSION_TIMEOUT_SEC	Integer	Minimum session timeout period, in seconds, that client is allowed to specify.
uaserverinterface.PROJECT_OPC_UA_MAX_SESSION_TIMEOUT_SEC	Integer	Maximum session timeout period, in seconds, that client is allowed to specify.
uaserverinterface.PROJECT_OPC_UA_TAG_CACHE_TIMEOUT_SEC	Integer	Increase the timeout to improve performance for clients that perform reads / writes on unregistered tags.
uaserverinterface.PROJECT_OPC_UA_BROWSE_TAG_PROPERTIES	YesNo	Return tag properties when a client browses the server address space.
uaserverinterface.PROJECT_OPC_UA_BROWSE_ADDRESS_HINTS	YesNo	Return device addressing hints when a client browses the server address space.
uaserverinterface.PROJECT_OPC_UA_MAX_DATA_QUEUE_SIZE	Integer	Maximum number of data change notifications queued per monitored item. Higher limits give the client more flexibility but can lead to higher memory usage.
uaserverinterface.PROJECT_OPC_UA_MAX_RETRANSMIT_QUEUE_SIZE	Integer	Maximum number of notifications in the republish queue the server allows per subscription. Higher limits use more memory but allow clients to retransmit older messages.
uaserverinterface.PROJECT_OPC_UA_MAX_NOTIFICATION_PER_PUBLISH	Integer	Maximum number of notifications the server sends per publish. Use larger values for fast and reliable connections.
thingworxinterface.ENABLED	YesNo	Enable the ThingWorx native interface.
thingworxinterface.HOSTNAME	String	"The hostname or IP address of the ThingWorx Platform instance.
thingworxinterface.PORT	Integer	The port used to connect to the platform instance, commonly 443 for secure connections.
thingworxinterface.RESOURCE	String	The endpoint URL of the platform hosting the websocket server, such as '/ThingWorx/WS'.
thingworxinterface.APPKEY	String	The application key used to authenticate; this is generated in the platform.
thingworxinterface.ALLOW_SELF_SIGNED_CERTIFICATE	YesNo	Enable to trust valid self-signed certificates presented by the server (less secure).
thingworxinterface.TRUST_ALL_CERTIFICATES	YesNo	Enable to trust all server certificates (include self-signed and invalid) and com-

Property Name	Type	Description
		pletely disable certificate validation. Do not use on a production system.
thingworxinterface.DISABLE_ENCRYPTION	YesNo	Disable SSL/TLS and allow connecting to an insecure endpoint. Do not use on a production system.
thingworxinterface.THING_NAME	String	The thing name presented to the platform.
thingworxinterface.PUBLISH_FLOOR_MSEC	Integer	The minimum rate that updates are sent to the platform. Set to zero to send updates as fast as possible.
thingworxinterface.LOGGING_ENABLED	YesNo	Enable ThingWorx Advanced Logging. When enabled, advanced log information is routed to the server event log.
thingworxinterface.LOG_LEVEL	Enumeration: Trace: 0 Info: 2 Warning: 3 Error: 4 Audit: 6	Determines that amount of information logged. Set to Trace to generate the most detailed output.
thingworxinterface.VERBOSE	YesNo	Determines the level of detail of each message logged. Set to Yes to add additional verbosity.
thingworxinterface.PROXY_ENABLED	YesNo	Enables ThingWorx proxy support.
thingworxinterface.PROXY_HOST	String	Specify the IP address or DNS name of the proxy server to connect.
thingworxinterface.PROXY_PORT	Integer	Specify the number of the TCP port used to connect to the proxy server.
thingworxinterface.PROXY_USERNAME	String	Enter the password authentication string for connecting to the ThingWorx server as the user specified.
thingworxinterface.PROXY_PASSWORD	String	Enter the password authentication string for connecting to the ThingWorx server as the user specified.

Example Request

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project
```

Example Response

```
{
 "PROJECT_ID": 3536816236,
 "common.ALLTYPES_DESCRIPTION": "",
 "servermain.PROJECT_TITLE": "",
 "servermain.PROJECT_TAGS_DEFINED": "121",
 "uaserverinterface.PROJECT_OPC_UA_ENABLE": true
 "uaserverinterface.PROJECT_OPC_UA_DIAGNOSTICS": false,
 "uaserverinterface.PROJECT_OPC_UA_ANONYMOUS_LOGIN": true,
 "uaserverinterface.PROJECT_OPC_UA_MAX_CONNECTIONS": 128,
 "uaserverinterface.PROJECT_OPC_UA_MIN_SESSION_TIMEOUT_SEC": 15,
 "uaserverinterface.PROJECT_OPC_UA_MAX_SESSION_TIMEOUT_SEC": 60,
 "uaserverinterface.PROJECT_OPC_UA_TAG_CACHE_TIMEOUT_SEC": 5,
 "uaserverinterface.PROJECT_OPC_UA_BROWSE_TAG_PROPERTIES": false,
 "uaserverinterface.PROJECT_OPC_UA_BROWSE_ADDRESS_HINTS": false,
 "uaserverinterface.PROJECT_OPC_UA_MAX_DATA_QUEUE_SIZE": 2,
 "uaserverinterface.PROJECT_OPC_UA_MAX_RETRANSMIT_QUEUE_SIZE": 10,
 "uaserverinterface.PROJECT_OPC_UA_MAX_NOTIFICATION_PER_PUBLISH": 65536,
 "thingworxinterface.ENABLED": false,
 "thingworxinterface.HOSTNAME": "hostname_or_ip",
 "thingworxinterface.PORT": 443,
 "thingworxinterface.RESOURCE": "/ThingWorx/WS",
 "thingworxinterface.APPKEY": "",
 "thingworxinterface.ALLOW_SELF_SIGNED_CERTIFICATE": false,
 "thingworxinterface.TRUST_ALL_CERTIFICATES": false,
 "thingworxinterface.DISABLE_ENCRYPTION": false,
 "thingworxinterface.THING_NAME": "ThingWorxKepwareEdge",
 "thingworxinterface.PUBLISH_FLOOR_MSEC": 1000,
 "thingworxinterface.LOGGING_ENABLED": false,
 "thingworxinterface.LOG_LEVEL": 3,
 "thingworxinterface.VERBOSE": false,
 "thingworxinterface.PROXY_ENABLED": false, "thingworxinterface.PROXY_HOST": "loc-
alhost", "thingworxinterface.PROXY_PORT": 3128, "thingworxinterface.PROXY_USERNAME":
"",
 "thingworxinterface.PROXY_PASSWORD": "" }
```

Project Properties — OPC UA

OPC Unified Architecture (UA) provides a platform independent interoperability standard. The OPC UA Project Properties group displays the current OPC UA settings in the server.

Server Interface

Enable: When enabled, the UA server interface is initialized and accepts client connections. When disabled, the remaining properties on this page are disabled.

Client Sessions

Allow anonymous login: This property specifies whether or not a user name and password are required to establish a connection. For security, the default setting is No to disallow anonymous access and require credentials to log in.

Tip: Additional users may be configured to access data without all the permissions associated with the administrator account. When the client supplies a password on connect, the server decrypts the password using the encryption algorithm defined by the security policy of the endpoint, then uses it to login.

Note: Users can login as the Administrator using the password set during the installation of ThingWorx Kepware Edge to login. Additional users may be configured to access data without all the permissions associated with the administrator account. When the client supplies a password on connect, the server decrypts the password using the encryption algorithm defined by the security policy of the endpoint, then uses it to login.

When the client supplies a password on connect, the server decrypts the password using the encryption algorithm defined by the security policy of the endpoint.

Max. connections: specify the maximum number of supported connections. The valid range is 1 to 128. The default setting is 128.

Minimum session timeout: specify the UA client's minimum timeout limit for establishing a session. Values may be changed depending on the needs of the application. The default value is 15 seconds.

Maximum session timeout: specify the UA client's maximum timeout limit for establishing a session. Values may be changed depending on the needs of the application. The default value is 60 seconds.

Tag cache timeout: specify the tag cache timeout. The valid range is 0 to 60 seconds. The default setting is 5 seconds.
Note: This timeout controls how long a tag is cached after a UA client is done using it. In cases where UA clients read / write to unregistered tags at a set interval, users can improve performance by increasing the timeout. For example, if a client is reading an unregistered tag every 5 seconds, the tag cache timeout should be set to 6 seconds. Since the tag does not have to be recreated during each client request, performance improves.

Browsing

Return tag properties: Enable to allow UA client applications to browse the tag properties available for each tag in the address space. This setting is disabled by default.

Return address hints: Enable to allows UA client applications to browse the address formatting hints available for each item. Although the hints are not valid UA tags, certain UA client applications may try to add them to the tag database. When this occurs, the client receives an error from the server. This may cause the client to report errors or stop adding the tags automatically. To prevent this from occurring, make sure that this property is disabled. This setting is disabled by default.

Monitored Items

Max. Data Queue Size: specify the maximum number of data notifications to be queued for an item. The valid range is 1 to 100. The default setting is 2.
Note: The data queue is used when the monitored item's update rate is faster than the subscription's publish rate. For example, if the monitored item update rate is 1 second, and a subscription publishes every 10 seconds, then 10 data notifications are published for the item every 10 seconds. Because queuing data consumes memory, this value should be limited when memory is a concern.

Subscriptions

Max. retransmit queue size: specify the maximum number of publishes to be queued per subscription. The valid range is 1 to 100. A value of zero disables retransmits. The default setting is 10.
Note: Subscription publish events are queued and retransmitted at the client's request. Because queuing consumes memory, this value should be limited when memory is a concern.

Max. notifications per publish: specify the maximum number of notifications per publish. The valid range is 1 to 65536. The default setting is 65536.
Note: This value may affect the connection's performance by limiting the size of the packets sent from the server to the client. In general, large values should be used for high-bandwidth connections and small values should be used for low-bandwidth connections.

Channel Properties — Configuration API

The following properties define a channel using the Configuration API service.

General Properties

`common.ALLTYPES_NAME` * Required parameter

Note: Changing this property causes the API endpoint URL to change.

`common.ALLTYPES_DESCRIPTION`

`servermain.MULTIPLE_TYPES_DEVICE_DRIVER` * Required parameter

Ethernet Communication Properties

`servermain.CHANNEL_ETHERNET_COMMUNICATIONS_NETWORK_ADAPTER_STRING`

Advanced Properties

`servermain.CHANNEL_NON_NORMALIZED_FLOATING_POINT_HANDLING`

Write Optimizations

`servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD`

`servermain.CHANNEL_WRITE_OPTIMIZATIONS_DUTY_CYCLE`

***See Also**: The server help system Configuration API Service section.*

Configuration API Service — Creating a Channel

To create a channel via the Configuration API service, only a minimum set of properties are required; all others are set to the default value. Once a channel is defined, its properties and settings are used by all devices assigned to that channel. The specific properties are dependent on the protocol or driver selected.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the channel endpoint.

The example below creates a channel named Channel1 that uses the Simulator driver on a server running on the local host.

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels
```

Body:

```
{
  “common.ALLTYPES_NAME”: “Channel1”,
  “servermain.MULTIPLE_TYPES_DEVICE_DRIVER”: “Simulator”
}
```

Refer to the driver specific help documentation to find out what properties are required to create a channel for that driver.

Configuration API Service — Updating a Channel

To update a property or collection of properties on a channel, a GET request must first be sent to the endpoint to be updated to get the Project ID.

For more information about the Project ID see the Concurrent Clients section.

In the example below, the channel being updated is Channel1.

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1
```

The GET request will return a JSON blob similar to the following.

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "Channel1",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
  "servermain.CHANNEL_UNIQUE_ID": 2154899492,
  "servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD": 2,
…
```

To update or change a channel property, a PUT request is sent to the channel with the Project ID and the new property value defined. In the following example, the channel name will change from Channel1 (from above) to Simulator.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1
```

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "Simulator"
}
```

Following the PUT, a GET can be sent to the channel’s endpoint to validate that the property changed. In this case, because the name was changed, the endpoint also changed and the GET request would be the following.

Note: Some properties are client restricted and cannot be changed when a client is connected.

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Simulator
```

The response from the GET request should show the property value has changed. The response to the GET above should look similar to the following:

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "Simulator",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
  "servermain.CHANNEL_UNIQUE_ID": 2154899492,
  "servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD": 2,
…
```

Configuration API Service — Removing Channel

To remove a channel, send a DELETE command to the channel endpoint to be removed. This causes the channel and all of its children to be removed.

In the example below, the channel Simulator will be removed.

Endpoint (DELETE):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Simulator
```

This can be verified by sending a GET to the removed endpoint. The server will respond with an error. It can also be verified with a GET to the "channels" endpoint; the removed channel will not be in the list of channels returned from the GET request.

Device Properties — Configuration API

The following properties define a device using the Configuration API service.

General Properties

`common.ALLTYPES_NAME` * Required parameter

`common.ALLTYPES_DESCRIPTION`

`servermain.DEVICE_CHANNEL_ASSIGNMENT`

`servermain.MULTIPLE_TYPES_DEVICE_DRIVER` * Required parameter

`servermain.DEVICE_MODEL` * Not required, but verify the default is acceptable

`servermain.DEVICE_ID_STRING` * Required parameter

`servermain.DEVICE_DATA_COLLECTION`

`servermain.DEVICE_SIMULATED`

Scan Mode

`servermain.DEVICE_SCAN_MODE`

`servermain.DEVICE_SCAN_MODE_RATE_MS`

`servermain.DEVICE_SCAN_MODE_RATE_MS`

`servermain.DEVICE_SCAN_MODE_PROVIDE_INITIAL_UPDATES_FROM_CACHE`

Auto Demotion

`servermain.DEVICE_AUTO_DEMOTION_ENABLE_ON_COMMUNICATIONS_FAILURES`

`servermain.DEVICE_AUTO_DEMOTION_DEMOTE_AFTER_SUCCESSIVE_TIMEOUTS`

`servermain.DEVICE_AUTO_DEMOTION_PERIOD_MS`

```
servermain.DEVICE_AUTO_DEMOTION_DISCARD_WRITES
```

Tag Generation

```
servermain.DEVICE_TAG_GENERATION_ON_STARTUP

servermain.DEVICE_TAG_GENERATION_DUPLICATE_HANDLING

servermain.DEVICE_TAG_GENERATION_GROUP

servermain.DEVICE_TAG_GENERATION_ALLOW_SUB_GROUPS
```

Tip: To Invoke Automatic Tag Generation, send a PUT with an empty body to the TagGeneration service endpoint on the device.

Note: All files in the user_data directory must be world readable or owned by the ThingWorx Kepware Edge user and group that were created during installation, by default tkedge.

See Also: *For more information, see Services help.*

Timing

```
servermain.DEVICE_CONNECTION_TIMEOUT_SECONDS

servermain.DEVICE_REQUEST_TIMEOUT_MILLISECONDS

servermain.DEVICE_RETRY_ATTEMPTS

servermain.DEVICE_INTER_REQUEST_DELAY_MILLISECONDS
```

See Also: *The server help system Configuration API Service section.*

Configuration API Service — Creating a Device

To create a device via the Configuration API service, only a minimum set of properties are required; all others are set to the default value. The specific properties are dependent on the protocol or driver selected.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the device endpoint under a channel.

The example below will create a device named Device1 under Channel1 that uses the Simulator driver on a server running on the local host.

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices
```

Body:

```
{
  “common.ALLTYPES_NAME”: “Device1”,
  “servermain.MULTIPLE_TYPES_DEVICE_DRIVER”: “Simulator”
}
```

Refer to the driver specific help documentation to find out what properties are required to create a device for that driver.

Configuration API Service — Updating a Device

To update a property or collection of properties on a device, a GET request must first be sent to the endpoint to be updated to get the Project ID.

*For more information about the Project ID, see the **Concurrent Clients** section.*

In the example below, the device being updated is Device1 under Channel1.

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices/Device1
```

The GET request will return a JSON blob similar to the following.

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "Device1",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
  "servermain.DEVICE_MODEL": 0,
  "servermain.DEVICE_UNIQUE_ID": <project_ID_from_GET>,
  "servermain.DEVICE_CHANNEL_ASSIGNMENT": "Channel1",
…
```

To update or change a device property a PUT request is sent to the device with the Project ID and the new property value defined. In the following example the device name will change from Device1 (from above) to Simulator.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices/Device1
```

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "Simulator"
}
```

Following the PUT, a GET can be sent to the device endpoint to validate that the property changed. In this case, because the name was changed, the endpoint also changed and the GET request would be the following.

Note: Some properties are client restricted and cannot be changed when a client is connected.

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices/Simulator
```

The response from the GET request will show the property value has changed. The response to the GET above should look similar to the following.

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "Simulator",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Simulator",
  "servermain.DEVICE_MODEL": 0,
  "servermain.DEVICE_UNIQUE_ID": <device_ID_from_GET>,
  "servermain.DEVICE_CHANNEL_ASSIGNMENT": "Channel1",
…
```

Configuration API Service — Removing a Device

To remove a device, send a DELETE to the device endpoint to be removed. This will cause the device and all of its children to be removed.

In the example below, the device Simulator will be removed.

Endpoint (DELETE):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices/Simulator
```

This can be verified by sending a GET to the removed endpoint. The server will respond with an error. It can also be verified with a get to the devices endpoint and the removed device will not be in the list of devices returned from the GET request.

Configuration API Service — Creating a Tag

To create a tag via the Configuration API service, only a minimum set of properties are required; all others are set to the default value. The specific properties are dependent on the protocol or driver selected.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the tags endpoint under a device.

The example below will create a tag named MyTag for address R5 under Channel1/Device1 that uses the Simulator driver on a server running on the local host.

Endpoint (POST):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tags
```

Body:

```
{
  “common.ALLTYPES_NAME”: “MyTag”,
  "servermain.TAG_ADDRESS": "R5"
}
```

Tags can also be created within a tag group. The process for adding a tag group is the same except the URL will change to include the tag_group endpoint and the group name.
In the following example, the tag group RampTags already exists and a tag named MyTag will be created under it with the address R5.

⬢ *For more information on creating a tag group, see* ***Creating a Tag Group*** *section.*

Endpoint (POST):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tag_group/RampTags/tags
```

Body:

```
{
  “common.ALLTYPES_NAME”: “MyTag”,
  "servermain.TAG_ADDRESS": "R5"
}
```

● *Refer to the driver specific help documentation to find out what properties are required to create a tag for that driver.*

Configuration API Service — Updating a Tag

To update a property or collection of properties on a tag, a GET request must first be sent to the endpoint to be updated to get the Project ID.

● *For more information about the Project ID see the **Concurrent Clients** section.*

In the example below, the tag being updated is MyTag under Channel1/Device1.

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tags/MyTag
```

The GET request will return a JSON blob similar to the following.

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "MyTag",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.TAG_ADDRESS": "R0005",
  "servermain.TAG_DATA_TYPE": 5,
  "servermain.TAG_READ_WRITE_ACCESS": 1,
  "servermain.TAG_SCAN_RATE_MILLISECONDS": 100,
…
```

To update or change a tag property, a PUT request is sent to the tag with the Project ID and the new property value defined.
In the following example, the tag name will change from MyTag (from above) to Tag1.

Endpoint (PUT):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tags/MyTag
```

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "Tag1"
}
```

Following the PUT a GET can be sent to the tag's endpoint to validate that the property changed. In this case, because the name was changed, the endpoint also changed and the GET request would be the following.

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tags/Tag1
```

The response from the GET request will show the property value has changed. The response to the GET above should look similar to the following.

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "Tag1",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.TAG_ADDRESS": "R0005",
  "servermain.TAG_DATA_TYPE": 5,
  "servermain.TAG_READ_WRITE_ACCESS": 1,
  "servermain.TAG_SCAN_RATE_MILLISECONDS": 100,
…
```

Configuration API Service — Removing a Tag

To remove a tag, send a DELETE to the tag's endpoint to be removed. This will cause the tag and all of its children to be removed.
In the example below, the tag Tag1 will be removed.

Endpoint (DELETE):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices/Device1/tags/Tag1
```

This can be verified by sending a GET to the removed endpoint. The server will respond with an error. It can also be verified with a get to the tags endpoint and the removed tag will not be in the list of tags returned from the GET request.

Configuration API Service — Creating a Tag Group

To create a tag group via the Configuration API service, only a group name is required.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the tag_groups endpoint under a device.

The example below will create a tag group named RampTags under Channel1/Device1 that uses the Simulator driver on a server running on the local host.

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices/Device1/tag_groups
```

Body:

```
{
  “common.ALLTYPES_NAME”: “RampTags”
}
```

Tag groups can have tags and more tag groups nested under them. *To add a Tag, see the **Creating a Tag** section.*

To nest a Tag Group within another group, another POST action is required to add the existing group name and the tag_groups endpoint to the end of the URL.

Continuing the example above, the new request would look like the following.

Endpoint (POST):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tag_groups/RampTags/tag_
groups
```

Body:

```
{
  “common.ALLTYPES_NAME”: “1-10”
}
```

Configuration API Service — Updating a Tag Group

To update a property or collection of properties on a tag, a GET request must first be sent to the endpoint to be updated to get the Project ID.

*For more information about the Project ID, see the **Concurrent Clients** section.*

In the example below, the tag group being updated is RampTags under Channel1/Device1.

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tag_groups/RampTags
```

The GET request will return a JSON blob similar to the following.

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "RampTags",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.TAGGROUP_LOCAL_TAG_COUNT": 0,
  "servermain.TAGGROUP_TOTAL_TAG_COUNT": 0,
  "servermain.TAGGROUP_AUTOGENERATED": false
}
```

To update or change a tag group property, a PUT request is sent to the tag group with the Project ID and the new property value defined.

In the following example, the tag group name will change from RampTags (from above) to RampGroup.

Endpoint (PUT):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tags/MyTag
```

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "RampGroup"
}
```

Following the PUT, a GET can be sent to the tag group endpoint to validate that the property changed. In this case, because the name was changed, the endpoint also changed and the GET request would be the following.

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/Channel1/devices/Device1/tag_groups/RampGroup
```

The response from the GET request will show the property value has changed. The response to the GET above should look similar to the following.

Body:

```
{
  "PROJECT_ID": <project_ID_from_GET>,
  "common.ALLTYPES_NAME": "RampTags",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.TAGGROUP_LOCAL_TAG_COUNT": 0,
  "servermain.TAGGROUP_TOTAL_TAG_COUNT": 0,
  "servermain.TAGGROUP_AUTOGENERATED": false
}
```

Configuration API Service — Removing a Tag Group

To remove a tag group, send a DELETE to the tag group endpoint to be removed. This will cause the tag group and all of its children to be removed. In the example below the tag group RampGroup will be removed.

Endpoint (DELETE):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/Channel1/devices/Device1/tag_groups/RampGroup
```

This can be verified by sending a GET to the removed endpoint. The server will respond with an error. It can also be verified with a get to the tag_groups endpoint and the removed tag group will not be in the list of tag groups returned from the GET request.

Configuration API Service — Property Validation Error Object

When making a POST request to create an object or making a PUT request to update an object or project properties, new values for those properties may be input as the body of the PUT or POST request to change the values. If there is a property validation error, two error objects appear. The first error object contains an error code and a message detailing why the error occurred. The second error object shows the same error code and error message in addition to an error property value, a description of that error property, and the line of input that created the error. The following example shows the error object of a POST request to create an object with a name that already exists.

Response Body:

```
{
   “property”: “common.ALLTYPES_NAME”,
   “description”: “The name “Channel1” is already used.”,
   “error_line”: 7,
   "code": 400,
   "message": "Validation failed on property common.ALLTYPES_NAME in object definition at line 7: The name 'Channel1' is already used."
}
```

Configuration API Service — User Management

The User Manager controls client access to the project's objects (which are the channels, devices, tags. etc.) and their corresponding functions. The User Manager allows permissions to be specified by user groups. For example, the User Manager can restrict user access to project tag data based on its permissions from the parent user group.

The User Manager has built-in groups each contain a built-in user. The default groups are Administrators, Server Users, Anonymous Clients. The default users in these groups are Administrator, Default User, Data Client. Users cannot rename or change the description fields of built-in user groups or users. Neither the default groups nor the default users can be disabled.

User Groups

Endpoint: https://<hostname_or_ip>:<port>/config/v1/admin/server_usergroups

Supported Actions

HTTP(S) Verb	Action
POST	Create the specified group
GET	Retrieves a list of all groups
DELETE	Removes the specified group and all of its users

Endpoint: https://<hostname_or_ip>:<port>/config/v1/admin/server_usergroups/<GroupName>

Supported Actions

HTTP(S) Verb	Action
GET	Retrieves the specified group
PUT	Updates the specified group
DELETE	Removes the specified user

Properties

Property Name	Type	Required	Description
common.ALLTYPES_NAME	String	Yes	Specify the identity of this object.
common.ALLTYPES_DESCRIPTION	String	No	Provide a brief summary of this object or its use.
libadminsettings.USERMANAGER_GROUP_ENABLED	Enable/Disable	No	The group's enabled-state takes precedence over the users enabled state.
libadminsettings.USERMANAGER_IO_TAG_READ	Enable/Disable	No	Allow/deny clients belonging to the group to access I/O tag data.
libadminsettings.USERMANAGER_IO_TAG_WRITE	Enable/Disable	No	Allow/deny clients belonging to the group to modify I/O tag data. Note: When USERMANAGER_IO_TAG_READ is set to false, this property is also set to false and disabled to prevent write-only tags.
libadminsettings.USERMANAGER_IO_TAG_DYNAMIC_ADDRESSING	Enable/Disable	No	Allow/deny clients belonging to the group to add items using dynamic addressing.
libadminsettings.USERMANAGER_SYSTEM_TAG_READ	Enable/Disable	No	Allow/deny clients belonging to the group to access system tag data.
libadminsettings.USERMANAGER_SYSTEM_TAG_WRITE	Enable/Disable	No	Allow/deny clients belonging to the group to modify system tag data. Note: When USERMANAGER_SYSTEM_TAG_READ is set to false, this property is also set to false and disabled to prevent write-only tags.

Property Name	Type	Required	Description
libadminsettings.USERMANAGER_INTERNAL_TAG_READ	Enable/Disable	No	Allow/deny clients belonging to the group to access internal tag data.
libadminsettings.USERMANAGER_INTERNAL_TAG_WRITE	Enable/Disable	No	Allow/deny clients belonging to the group to modify internal tag data. Note: When USERMANAGER_INTERNAL_TAG_READ is set to false, this property is also set to false and disabled to prevent write-only tags.
libadminsettings.USERMANAGER_SERVER_MANAGE_LICENSES	Enable/Disable	No	Allow/deny users belonging to the group to access the license manager.
libadminsettings.USERMANAGER_SERVER_MODIFY_SERVER_SETTINGS	Enable/Disable	No	Allow/deny users belonging to the group to access this property sheet.
libadminsettings.USERMANAGER_SERVER_DISCONNECT_CLIENTS	Enable/Disable	No	Allow/deny users belonging to the group to take action that can cause data clients to be disconnected.
libadminsettings.USERMANAGER_SERVER_RESET_EVENT_LOG	Enable/Disable	No	Allow/deny users belonging to the group to clear all logged event messages.
libadminsettings.USERMANAGER_SERVER_OPCUA_DOTNET_CONFIGURATION	Enable/Disable	No	Allow/deny users belonging to the group to access the OPC UA or XI configuration manager.
libadminsettings.USERMANAGER_SERVER_CONFIG_API_LOG_ACCESS	Enable/Disable	No	Allow/deny users belonging to the group to access the Configuration API Transaction Log.
libadminsettings.USERMANAGER_SERVER_REPLACE_RUNTIME_PROJECT	Enable/Disable	No	Allow/deny users belonging to the group to replace the running project.
libadminsettings.USERMANAGER_BROWSE_BROWSENAMESPACE	Enable/Disable	No	Allow/deny clients belonging to the user group to browse the project namespace.

Project Permissions

Endpoint: https://<hostname_or_ip>:<port>/config/v1/admin/server_usergroups/<GroupName>/project_permissions

Supported Actions

HTTP(S) Verb	Action
GET	Retrieves a list of all project permissions

Child Endpoints

Properties

Endpoint	Description
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Alias	Configure default 'Servermain Alias' access permissions for the selected user group.
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Channel	Configure default 'Servermain Channel' access permissions for the selected user group.
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Device	Configure default 'Servermain Device' access permissions for the selected user group.
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Meter Order	Configure default 'Servermain Meter Order' access permissions for the selected user group. **Note**: Add and delete properties are disabled for this endpoint.
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Phone Number	Configure default 'Servermain Phone Number' access permissions for the selected user group.
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Phone Priority	Configure default 'Servermain Phone Priority' access permissions for the selected user group. **Note**: Add and delete properties are disabled for this endpoint.
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Project	Configure default 'Servermain Project' access permissions for the selected user group. **Note**: Add and delete properties are disabled for this endpoint.
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Tag	Configure default 'Servermain Tag' access permissions for the selected user group.
/config/v1/admin/server_usergroups/<GroupName>/project_permissions/Servermain Tag Group	Configure default 'Servermain Tag Group' access permissions for the selected user group.

Endpoint: https://<hostname_or_ip>:<port>/config/v1/admin/server_usergroups/<GroupName>/project_permissions/<PermissionName>

Supported Actions

HTTP(S) Verb	Action
GET	Retrieves the specified project permission
PUT	Updates the specified project permission

Properties

Property Name	Type	Description
common.ALLTYPES_NAME	String	Specify the identity of this object.
common.ALLTYPES_DESCRIPTION	String	Provide a brief summary of this object or its use.
libadminsettings.USERMANAGER_	Enable/Disable	Allow/deny users belonging to the group to

Property Name	Type	Description
PROJECTMOD_ADD		add this type of object.
libadminsettings.USERMANAGER_PROJECTMOD_EDIT	Enable/Disable	Allow/deny users belonging to the group to edit this type of object.
libadminsettings.USERMANAGER_PROJECTMOD_DELETE	Enable/Disable	Allow/deny users belonging to the group to delete this type of object.

Users

Endpoint: https://<hostname_or_ip>:<port>/config/v1/admin/server_users

Supported Actions

HTTP(S) Verb	Action
POST	Create the specified user
GET	Retrieves a list of all users

Endpoint: https://<hostname_or_ip>:<port>/config/v1/admin/server_users/<UserName>

Supported Actions

HTTP(S) Verb	Action
GET	Retrieves the specified user
PUT	Updates the specified user

Properties

Property Name	Type	Required	Description
common.ALLTYPES_NAME	String	Yes	Specify the identity of this object.
common.ALLTYPES_DESCRIPTION	String	No	Provide a brief summary of this object or its use.
libadminsettings.USERMANAGER_USER_GROUPNAME	String	Yes	The name of the parent group.
libadminsettings.USERMANAGER_USER_ENABLED	Enable/Disable	No	The group's enabled-state takes precedence over the users enabled state.
libadminsettings.USERMANAGER_USER_PASSWORD	Password	No	The user's password. This is case-sensitive. The password must be at least 14 characters and no more than 512 characters. Passwords should include a mix of uppercase and lowercase letters, numbers, and special characters. Avoid well-known, easily guessed, or common passwords.

Note: If there are errors when writing to read / write system tags, verify that the authenticated user has the appropriate permissions.

Configuration API Service — Creating a User

To create a user via the Configuration API service, only a minimum set of properties are required; all others are set to the default value.

Only members of the Administrators group can create users.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the server_users endpoint.

The example below creates a user named User1 that is a member of the server Administrators user group:

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/admin/server_users
```

Body:

```
{
   "common.ALLTYPES_NAME": "User1",
   "libadminsettings.USERMANAGER_USER_GROUPNAME": "Administrators",
   "libadminsettings.USERMANAGER_USER_PASSWORD": "<Password>"
}
```

The Administrator user account password cannot be reset, but additional administrative users can be added to the Administrator user group. Best practices suggest each user with administrative access be assigned unique accounts and passwords to ensure audit integrity and continual access through role and staff changes.

The product Administrator password must be at least 14 characters and no more than 512. Passwords should include a mix of uppercase and lowercase letters, numbers, and special characters. Choose a strong unique password that avoids well-known, easily guessed, or common passwords.

Configuration API Service — Creating a User Group

To create a group via the Configuration API service, only a minimum set of properties are required; all others are set to the default value. Once a user group is defined, its permissions are used by all users assigned to that user group.

Only members of the Administrators group can create user groups.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the server_usergroups endpoint.

The example below creates a user group named Operators:

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/admin/server_usergroups
```

Body:

```
{
  “common.ALLTYPES_NAME”: “Operators”,
}
```

Configuration API Service — Updating a User

To update a user via the Configuration API service, provide new values for the properties that require updating.

Only members of the Administrators group can update users.

There is no PROJECT_ID field for users.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the server_users/<username> endpoint.

The example below updates the user named User1 to add a description and move it to a different user group:

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/admin/server_users/User1
```

Body:

```
{
"common.ALLTYPES_DESCRIPTION": "The user account of User1", "libadminsettings.USERMANAGER_USER_GROUPNAME": "Operators"
}
```

Configuration API Service — Updating a User Group

To edit a user group via the Configuration API service, provide new values for the properties that require updating.

Only members of the Administrators group can update user groups.

There is no PROJECT_ID field for user groups.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a PUT request to the server_usergroups/<groupname> endpoint.

The example below updates the user group named Operators to have permissions to modify server settings, cause clients to be disconnected, and loading new runtime projects; it also updates the description of the group:

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/admin/server_usergroups/Operators
```

Body:

```
{
   "common.ALLTYPES_DESCRIPTION": "User group for standard operators",
   "libadminsettings.USERMANAGER_SERVER_MODIFY_SERVER_SETTINGS": true,
   "libadminsettings.USERMANAGER_SERVER_DISCONNECT_CLIENTS": true,
   "libadminsettings.USERMANAGER_SERVER_REPLACE_RUNTIME_PROJECT": true
}
```

Note: Group permissions for the administrator group are locked and cannot be modified by any user to prevent an administrator from accidentally disabling a permission that could prevent administrators from modifying any user permissions. Only users in the Administrator group can modify the permissions for other groups.

Configuration API Service — Configuring User Group Project Permissions

All user groups contain a collection of project permissions. Each project permission corresponds to a specific permission applied when interacting with objects in the project. All permissions are always present under a user group (and therefore cannot be created nor deleted). An individual project permission can be granted or denied by updating that specific project permission under the desired User Group.

Only members of the Administrators group can update a user group's project permissions.

There is no PROJECT_ID field for project permissions.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a PUT request to the project_permissions/<permission_name> endpoint.

The example below updates the user-created user group named Operators to grant permission to users of that group to add, edit, and delete channels:

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/admin/server_usergroups/Operators/project_
permissions/Servermain Channel
```

Body:

```
{
   "libadminsettings.USERMANAGER_PROJECTMOD_ADD": true,
   "libadminsettings.USERMANAGER_PROJECTMOD_EDIT": true,
   "libadminsettings.USERMANAGER_PROJECTMOD_DELETE": true
}
```

Configuration API Service — Configuring Licensing Server

Parameters configuring the Licensing Server connection as well as various logging parameters, such as the Event Log are configured under the admin endpoint.

Note: There is no PROJECT_ID field for admin permissions.

Endpoint:

```
https://<hostname_or_ip>:<port>/config/v1/admin/
```

Supported Actions

HTTP(s) Verb	Action
GET	Retrieves a list of admin properties
PUT	Updates the specified admin properties

Properties

Name	Type	Default	Description
libadminsettings.LICENSING_SERVER_PORT	Integer	7070	The port number used to connect to license server for non-TLS connections
libadminsettings.LICENSING_SERVER_NAME	String	" "	Host name or IP address for the license server (character limit is 63 characters)

Name	Type	Default	Description
libadminsettings.LICENSING_SERVER_ENDPOINT	String	fne/bin/capability	URL endpoint for the license server
libadminsettings.LICENSING_SERVER_ENABLE	Enable/Disable	false	Enable the connection to the license server
libadminsettings.LICENSING_CHECK_PERIOD_MINS	Integer	5	Time in minutes between checks of the license state. A license check occurs whenever a feature in use requires a license until it successfully borrows one.
libadminsettings.LICENSING_SERVER_SSL_PORT	Integer	1443	The port number used to connect to License Server for TLS connections
libadminsettings.LICENSING_SERVER_ALLOW_INSECURE_COMMS	Enable/Disable	false	Enable an insecure (non-TLS) connection to the License Server
libadminsettings.LICENSING_SERVER_ALLOW_SELF_SIGNED_CERTS	Enable/Disable	false	Enable use of self-signed certificates when establishing a TLS connection to the license server. Self-signed certificates are not secure and should only be used for testing.
libadminsettings.LICENSING_CLIENT_ALIAS	String	" "	User-specified name used for requesting license from license server. Character limit is 63 characters.

Configuration API Service — OPC UA Endpoint

While the majority of the OPC UA configuration is located under the Projects endpoint, the ua-endpoints are configured under the admin endpoint:

See Also: *Project Properties — OPC UA*

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/admin/ua_endpoints
```

Supported Actions

HTTP(S) Verb	Action
GET	Retrieves a list of all UA endpoint objects
POST	Creates a new UA endpoint

Endpoint:

```
https://<hostname_or_ip>:<port>/config/v1/admin/ua_endpoints/<endpointName>
```

Supported Actions:

HTTP(S) Verb	Action
GET	Retrieves the specified UA endpoint
PUT	Updates the specified UA endpoint

Properties

Name	Type	Required	Default	Description
common.ALLTYPES_NAME	String	Yes	NA	Specifies the identity of this object
common.ALLTYPES_DESCRIPTION	String	No	""	Lists available network adapters found on the system. Adapters without assigned IP address are listed as disconnected.
libadminsettings.UACONFIGMANAGER_ENDPOINT_ENABLE	Enable/Disable	No	True	Defines if the endpoint is enabled or disabled
libadminsettings.UACONFIGMANAGER_ENDPOINT_ADAPTER	String	No	"Default"	Specifies the network adapter to which the endpoint will be bound. A list of network adapters installed on the system is provided in the endpoint Description property. The “Default” adapter indicates that the endpoint can bind to any adapter.

Name	Type	Required	Default	Description
				Note: Network adapters that do not have a valid IPv4 address can be used for configuring a UA Endpoint; however, an endpoint is only used when there is a valid IPv4 address during startup. The server needs to be reinitialized for endpoint configurations to be refreshed after configuration changes are made to the host's network adapters.
libadminsettings.UACONFIGMANAGER_ENDPOINT_PORT	Integer	No	49330	The port number to which the endpoint will be bound
libadminsettings.UACONFIGMANAGER_ENDPOINT_URL	String	No	""	The endpoint URL (READONLY). The property value is generated based on the selected network adapter and port property. **Note**: The property is blank when the specified network adapter does not have a valid IPv4 address.
libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_NONE	Enable/Disable	No	False	The accepted endpoint security policy: **None: Endpoint accepts insecure connections** **Note**: {Insecure} This setting is insecure and not recommended.
libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256_SHA256	Enum	No	2	The accepted endpoint security policy: **BASIC256_SHA256**: Endpoint accepts BASIC256_SHA256 encrypted connections

Name	Type	Required	Default	Description

				The value determines the supported message mode or disabled if no message mode is selected:
				Enum=Disabled:0
				Enum=Sign:1
				Enum=Sign and Encrypt:2
				Enum=Sign; Sign and Encrypt:3

libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC128_RSA15	Enum	No	0	The accepted endpoint security policy:
				BASIC128_RSA15: Endpoint accepts BASIC128_RSA encrypted connections.
				The value determines the supported message mode or disabled if no message mode is selected:
				Enum=Disabled:0
				Enum=Sign:1
				Enum=Sign and Encrypt:2
				Enum=Sign; Sign and Encrypt:3
				Note: {Deprecated}
				This security policy is deprecated.

libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256	Enum	No	0	The accepted endpoint security policy:
				BASIC256: Endpoint accepts BASIC256 encrypted connections
				The value determines the supported message mode or disabled if no message mode is selected:
				Enum=Disabled:0
				Enum=Sign:1
				Enum=Sign and Encrypt:2
				Enum=Sign; Sign and Encrypt:3
				Note: {Deprecated}

Name	Type	Required	Default	Description
				This security policy is deprecated.

Note: A maximum of 100 OPC UA endpoints may be configured on a single instance of ThingWorx Kepware Edge.

Configuration API Service — Creating a UA Endpoint

To create a UA endpoint via the Configuration API service, only a minimum set of properties are required; all others are set to their default value.

To create a new UA endpoint, use a REST-based API tool such as Postman, Insomnia, or Curl and make a POST request to the admin/ua_endpoints endpoint.

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/admin/ua_endpoints
```

Body:

```
{
  "common.ALLTYPES_NAME": "Endpoint1"
}
```

Configuration API Service — Updating a UA Endpoint

To update a UA endpoint via the Configuration API service, provide new values for the properties that require updating.

Using a REST-based API tool such as Postman, Insomnia, or Curl; make a POST request to the ua_endpoints/<endpoint> endpoint.

The example below updates the endpoint named Endpoint1 with a new port number and security policy:

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/admin/ua_endpoints/Endpoint1
```

Body:

```
{
"libadminsettings.UACONFIGMANAGER_ENDPOINT_PORT": 49321,
"libadminsettings.UACONFIGMANAGER_ENDPOINT_SECURITY_BASIC256": 1
}
```

Configuration API Service — Removing a UA Endpoint

To delete an existing UA endpoint, make a DELETE request to the ua_endpoints/<endpoint_name> endpoint. A request body is not required:

Endpoint (DELETE):

```
https://<hostname_or_ip>:<port>/config/v1/admin/ua_endpoints/Endpoint1
```

Body:

```
{
 }
```

Connecting with an OPC UA Client Using UaExpert

An application like Unified Automation's UaExpert can be used to verify the flow of data from devices through ThingWorx Kepware Edge.

The UaExpert tool is designed to be a general-purpose OPC UA test client; it is not meant for production. Below is a walk-through of creating a secure user with specific data access rights to read and write tags.

Default OPC UA Server Settings

- URL: opc.tcp://<hostname>:<port>
- Port: 49330
- Security Policies: Basic256Sha256
- Authentication: (Enabled by default)
- Server Interface Enabled: True

Creating a User Group and User with Read / Write / Browse Access

1. Install ThingWorx Kepware Edge with default settings.
2. Add a new user group with data access and browse permissions via the Config API:

   Endpoint (POST):

   ```
   https://<hostname>:<port>/config/v1/admin/server_usergroups
   ```

   Body:

   ```
   {
   "common.ALLTYPES_NAME": "Group1",
   "libadminsettings.USERMANAGER_GROUP_ENABLED": true,
   "libadminsettings.USERMANAGER_IO_TAG_READ": true,
   "libadminsettings.USERMANAGER_IO_TAG_WRITE": true,
   "libadminsettings.USERMANAGER_BROWSE_BROWSENAMESPACE": true
   }
   ```

3. Add a new user with a password to the group created in above.

   Endpoint (POST):

   ```
   https://<hostname>:<port>/config/v1/admin/server_users
   ```

   Body:

   ```
   {
   "common.ALLTYPES_NAME": "User1",
   "libadminsettings.USERMANAGER_USER_GROUPNAME": "Group1",
   "libadminsettings.USERMANAGER_USER_ENABLED": true,
   "libadminsettings.USERMANAGER_USER_PASSWORD": "<insert_password>"
   }
   ```

Adding Server Connection to UaExpert

1. Download, install, and launch UaExpert from Unified Automation.
2. Select the **Server | Add** drop-down menu option.

3. In the **Add Server** configuration window, double-click the **Add Server** option located under **Custom Discovery**.

4. Enter the URL and port for the machine to connect. For example: “opc.tcp://<hostname>:49330”.

5. A new server connection is added in the Custom Discovery group.

6. Expand the new server connection for a list of valid endpoints. These are the available security options for the server. In this example, only one option is available.

7. Choose the **Basic256Sha256 – Sign & Encrypt** security option.

8. Set the user name and password using the settings used in the creation of the user above.

9. Check the **Store** checkbox to save the password or leave it unchecked and to be prompted for a password when connecting to the server.

10. Click **OK** to close the window.

11. Verify that "ThingWorxKepwareEdge/UA" appears under Servers.

12. Right-click on the server and select **Connect**.

13. A certificate validation window appears.

14. Click **Trust Server Certificate** for the client to trust the ThingWorxKepwareEdge/UA server.

15. Click **Continue**. There is an error until the server trusts the client certificate.

16. To trust the client certificate on the server, these instructions use the **edge_admin** tool *(see the server help for other methods)*.

17. The client certificate’s thumbprint is required to trust it. To get the thumbprint, use the edge_admin tool to list the certificates in the UA Server trust store:

    ```
    $ <installation_directory>/edge_admin manage-truststore --list uaserver
    ```

18. The output of the list shows a thumbprint, a status, and a common name of the certificate.
    The UaExpert certificate will be Rejected. Use the thumbprint to trust the certificate.

    ```
    $ <installation_directory>/edge_admin manage-truststore --trust=
    <certificate_thumbprint> uaserver
    ```

19. List the certificates of the UA Server to verify that the certificate is now trusted.

20. In UaExpert, right-click on the server and click **Connect**. The connection should succeed and the Address Space window in the lower right pane should be populated, which enables browsing for and adding tags.

21. Add a tag in the data access view to verify that the user has read access.

22. Change the value of the tag to verify that the user has write access.

Event Log Messages

The following information concerns messages posted to the Event Log. Server help contains many common messages, so should also be searched. Generally, the type of message (informational, warning) and troubleshooting information is provided whenever possible.

● *Please refer to the Running in a Container for information about additional Event Log features using ThingWorx Kepware Edge in a container.*

The Config API SSL certificate contains a bad signature.

Error Type:

Error

The Config API is unable to load the SSL certificate.

Error Type:

Error

Unable to start the Config API Service. Possible problem binding to port.

Error Type:

Error

Possible Cause:

The HTTP or HTTPS port specified in the Config API settings is already bound by another application.

Possible Solution:

Change the configuration of the Config API or blocking application to use a different port, or stop the application blocking the port.

The Config API SSL certificate has expired.

Error Type:

Warning

The Config API SSL certificate is self-signed.

Error Type:

Warning

The configured version of TLS for the Configuration API is no longer considered secure. It is recommended that only TLS 1.2 or higher is used.

Error Type:

Warning

Configuration API started without SSL on port <port number>.

Error Type:

Informational

Configuration API started with SSL on port <port number>.

Error Type:

Informational

The <name> device driver was not found or could not be loaded.

Error Type:

Error

Possible Cause:

1. If the project has been moved from one PC to another, the required drivers may have not been installed yet.
2. The specified driver may have been removed from the installed server.
3. The specified driver may be the wrong version for the installed server version.

Possible Solution:

1. Re-run the server install and add the required drivers.
2. Re-run the server install and re-install the specified drivers.
3. Ensure that a driver has not been placed in the installed server directory (which is out of sync with the server version).

Unable to load the '<name>' driver because more than one copy exists ('<name>' and '<name>'). Remove the conflicting driver and restart the application.

Error Type:

Error

Possible Cause:

Multiple versions of the driver DLL exist in the driver's folder in the server.

Possible Solution:

1. Re-run the server install and re-install the specified drivers.
2. Contact Technical support and verify the correct version. Remove the driver that is invalid and restart the server and load the project.

Invalid project file.

Error Type:

Error

Unable to add channel due to driver-level failure.

Error Type:

Error

Possible Cause:

Attempt failed due to issues in the driver.

Possible Solution:

Refer to the additional messages about the driver error and correct related issues.

Unable to add device due to driver-level failure.

Error Type:

Error

Possible Cause:

Attempt failed due to issues in the driver.

Possible Solution:

Refer to the additional messages about the driver error and correct related issues.

Version mismatch.

Error Type:

Error

Unable to load project <name>:

Error Type:

Error

Possible Cause:

1. The project was created using a version of the server that contained a feature or configuration that has been obsoleted and no longer exists in the server that is trying to load it.
2. The project was created in a server version that is not compatible with the version trying to load it.
3. The project file is corrupt.

Possible Solution:

Save project as JSON(V6), remove the unsupported feature that is defined in the project file and then save and load the updated project file into the server that is trying to load it.

Note:

Every attempt is made to ensure backwards compatibility in the server so that projects created in older versions may be loaded in newer versions. However, since new versions of the server and driver may have properties and configurations that do not exist in older versions, it may not be possible to open or load an older project in a newer version.

Unable to back up project file to '<path>' [<reason>]. The save operation has been aborted. Verify the destination file is not locked and has read/write access. To continue to save this project without a backup, deselect the backup option under Tools | Options | General and re-save the project.

Error Type:

Error

Possible Cause:

1. The destination file may be not locked by another application.
2. The destination file or the folder where it is located does not allow read/write access.

Possible Solution:

1. Ensure that the destination file is not locked by another application, unlock the file, or close the application.
2. Ensure that the destination file and with the folder where it is located allow read and write access.

<feature name> was not found or could not be loaded.

Error Type:

Error

Possible Cause:

The feature is not installed or is not in the expected location.

Possible Solution:

Re-run the server install and select the specified feature for installation.

Unable to save project file <name>:

Error Type:

Error

Device discovery has exceeded <count> maximum allowed devices. Limit the discovery range and try again.

Error Type:

Error

<feature name> is required to load this project.

Error Type:

Error

Unable to load the project due to a missing object. | Object = '<object>'.

Error Type:

Error

Possible Cause:

Editing the JSON project file may have left it in an invalid state.

Possible Solution:

Revert any changes made to the JSON project file.

Invalid Model encountered while trying to load the project. | Device = '<device>'.

Error Type:

Error

Possible Cause:

The specified device has a model that is not supported in this version of the server.

Possible Solution:

Open this project with a newer version of the server.

Cannot add device. A duplicate device may already exist in this channel.

Error Type:

Error

Auto-generated tag '<tag>' already exists and will not be overwritten.

Error Type:

Warning

Possible Cause:

Although the server is regenerating tags for the tag database, it has been set not to overwrite tags that already exist.

Possible Solution:

If this is not the desired action, change the setting of the "On Duplicate Tag" property for the device.

Unable to generate a tag database for device '<device>'. The device is not responding.

Error Type:

Warning

Possible Cause:

1. The device did not respond to the communications request.
2. The specified device is not on, not connected, or in error.

Possible Solution:

1. Verify that the device is powered on and that the PC is on (so that the server can connect to it).
2. Verify that all cabling is correct.
3. Verify that the device IDs are correct.
4. Correct the device failure and retry the tag generation.

Unable to generate a tag database for device '<device>':

Error Type:

Warning

Possible Cause:

The specified device is not on, not connected, or in error.

Possible Solution:

Correct the device failure and retry the tag generation.

Auto generation produced too many overwrites, stopped posting error messages.

Error Type:

Warning

Possible Cause:

1. To keep from filling the error log, the server has stopped posting error messages on tags that cannot be overwritten during automatic tag generation.
2. Reduce the scope of the automatic tag generation or eliminate problematic tags.

Failed to add tag '<tag>' because the address is too long. The maximum address length is <number>.

Error Type:

Warning

Unable to use network adapter '<adapter>' on channel '<name>'. Using default network adapter.

Error Type:

Warning

Possible Cause:

The network adapter specified in the project does not exist on this PC. The server uses the default network adapter.

Possible Solution:

Select the network adapter to use for the PC and save the project.

See Also:

Channel Properties - Network Interface

Rejecting attempt to change model type on a referenced device '<channel device>'.

Error Type:

Warning

Validation error on '<tag>': <error>.

Error Type:

Warning

Possible Cause:

An attempt was made to set invalid parameters on the specified tag.

Unable to load driver DLL '<name>'.

Error Type:

Warning

Possible Cause:

The specified driver could not be loaded when the project started.

Possible Solution:

1. Verify the version of the installed driver. Check the website to see if the driver version is correct for the server version installed.
2. If the driver corrupted, delete it and re-run the server install.

Note:

This problem is usually due to corrupted driver DLLs or drivers that are not compatible with the server version.

Validation error on '<tag>': Invalid scaling parameters.

Error Type:

Warning

Possible Cause:

An attempt was made to set invalid scaling parameters on the specified tag.

See Also:

Tag Properties - Scaling

Device '<device>' has been automatically demoted.

Error Type:

Warning

Possible Cause:

Communications with the specified device failed. The device has been demoted from the poll cycle.

Possible Solution:

1. If the device fails to reconnect, investigate the reason behind the communications loss and correct it.
2. To stop the device from being demoted, disable Auto-Demotion.

See Also:

Auto-Demotion

Unable to load plug-in DLL '<name>'.

Error Type:

Warning

Possible Cause:

The specified plug-in could not be loaded when the project started.

Possible Solution:

1. Verify the version of the plug-in installed. Check the website to see if the plug-in version is compatible with the server installed. If not, correct the server or re-run the server install.
2. If the plug-in is corrupt, delete it and then re-run the server install.

Note:

This problem is usually due to corrupted plug-in DLLs or plug-ins that are not compatible with the server version.

Unable to load driver DLL '<name>'. Reason:

Error Type:

Warning

Possible Cause:

The specified plug-in could not be loaded when the project started.

Possible Solution:

1. Verify the version of the plug-in installed. Check the website to see if the plug-in version is compatible with the server installed. If not, correct the server or re-run the server install.
2. If the plug-in is corrupt, delete it and then re-run the server install.

Unable to load plug-in DLL '<name>'. Reason:

Error Type:

Warning

Possible Cause:

The specified plug-in could not be loaded when the project started.

Possible Solution:

1. Verify the version of the plug-in installed. Check the website to see if the plug-in version is compatible with the server installed. If not, correct the server or re-run the server install.
2. If the plug-in is corrupt, delete it and then re-run the server install.

The specified network adapter is invalid on channel '%1' | Adapter = '%2'.

Error Type:

Warning

Possible Cause:

The network adapter specified in the project does not exist on this PC.

Possible Solution:

Select the network adapter to use for the PC and save the project.

See Also:

Channel Properties - Network Interface

No tags were created by the tag generation request. See the event log for more information.

Error Type:

Warning

Possible Cause:

The driver produced no tag information but declined to provide a reason why.

Possible Solution:

Event log may contain information that will help troubleshoot the issue.

<Product> device driver loaded successfully.

Error Type:

Informational

Starting <name> device driver.

Error Type:

Informational

Stopping <name> device driver.

Error Type:

Informational

<Product> device driver unloaded from memory.

Error Type:

Informational

Simulation mode is enabled on device '<device>'.

Error Type:

Informational

Simulation mode is disabled on device '<device>'.

Error Type:

Informational

Attempting to automatically generate tags for device '<device>'.

Error Type:

Informational

Completed automatic tag generation for device '<device>'.

Error Type:

Informational

A client application has enabled auto-demotion on device '<device>'.

Error Type:

Informational

Possible Cause:

A client application connected to the server has enabled or disabled Auto Demotion on the specified device.

Possible Solution:

To restrict the client application from doing this, disable its ability to write to system-level tags through the User Manager.

See Also:

User Manager

Data collection is enabled on device '<device>'.

Error Type:

Informational

Data collection is disabled on device '<device>'.

Error Type:

Informational

Object type '<name>' not allowed in project.

Error Type:

Informational

Created backup of project '<name>' to '<path>'.

Error Type:

Informational

Device '<device>' has been auto-promoted to determine if communications can be re-established.

Error Type:

Informational

Failed to load library: <name>.

Error Type:

Informational

Failed to read build manifest resource: <name>.

Error Type:

Informational

A client application has disabled auto-demotion on device '<device>'.

Error Type:

Informational

Tag generation results for device '<device>'. | Tags created = <count>.

Error Type:

Informational

Tag generation results for device '<device>'. | Tags created = <count>, Tags overwritten = <count>.

Error Type:

Informational

Tag generation results for device '<device>'. | Tags created = <count>, Tags not overwritten = <count>.

Error Type:

Informational

Access to object denied. | User = '<account>', Object = '<object path>', Permission =

Error Type:

Security

User moved from user group. | User = '<name>', Old group = '<name>', New group = '<name>'.

Error Type:

Security

User group has been created. | Group = '<name>'.

Error Type:

Security

User added to user group. | User = '<name>', Group = '<name>'.

Error Type:

Security

User group has been renamed. | Old name = '<name>', New name = '<name>'.

Error Type:

Security

Permissions definition has changed on user group. | Group = '<name>'.

Error Type:

Security

User has been renamed. | Old name = '<name>', New name = '<name>'.

Error Type:

Security

User has been disabled. | User = '<name>'.

Error Type:

Security

User group has been disabled. | Group = '<name>'.

Error Type:

Security

User has been enabled. | User = '<name>'.

Error Type:

Security

User group has been enabled. | Group = '<name>'.

Error Type:

Security

Password for user has been changed. | User = '<name>'.

Error Type:

Security

The endpoint '<url>' has been added to the UA Server.

Error Type:

Security

The endpoint '<url>' has been removed from the UA Server.

Error Type:

Security

The endpoint '<url>' has been disabled.

Error Type:

Security

The endpoint '<url>' has been enabled.

Error Type:

Security

User has been deleted. | User = '<name>'.

Error Type:

Security

Group has been deleted. | Group = '<name>'.

Error Type:

Security

Missing server instance certificate '<cert location>'. Please use the OPC UA Configuration Manager to reissue the certificate.

Error Type:

Error

Failed to import server instance cert: '<cert location>'. Please use the OPC UA Configuration Manager to reissue the certificate.

Error Type:

Error

Possible Cause:

1. The file containing the server instance certificate does not exist or is inaccessible.
2. Certificate decryption failed.

Possible Solution:

1. Verify the file references a valid instance certificate to which the user has permissions.
2. Import a new certificate.
3. Re-issue the certificate to refresh the encryption.

The UA server certificate is expired. Please use the OPC UA Configuration Manager to reissue the certificate.

Error Type:

Error

Possible Cause:

The validity period of the certificate is before the current system date.

Possible Solution:

1. Import a non-expired certificate.
2. Re-issue the certificate to generate a new non-expired certificate.

A socket error occurred listening for client connections. | Endpoint URL = '<endpoint URL>', Error = <error code>, Details = '<description>'.

Error Type:

Error

Possible Cause:

The endpoint socket returned an error while listening for client connections.

Possible Solution:

Note the details in the error message to diagnose the problem.

The UA Server failed to register with the UA Discovery Server. | Endpoint URL: '<endpoint url>'.

Error Type:

Error

Possible Cause:

1. The UA server endpoint URL and the security policy are not supported in the UA Discovery Server.
2. The attempt to register the UA Server with the UA Discovery Server could not complete in the expected manner.

Possible Solution:

Verify the UA Server endpoint URL and the security policy with the UA Discovery Server endpoints.

Unable to start the UA server due to certificate load failure.

Error Type:

Error

Possible Cause:

1. The UA Server application instance certificate validity period occurs before the current system date.
2. The file containing the server instance certificate does not exist or is inaccessible.
3. Certificate decryption failed.

Possible Solution:

1. Import a non-expired certificate.
2. Re-issue the certificate to generate a new non-expired certificate.
3. Verify the file references a valid instance certificate to which the user has permissions.
4. Re-issue the certificate to refresh the encryption.

Failed to load the UA Server endpoint configuration.

Error Type:

Error

Possible Cause:

The endpoint configuration file is corrupt or doesn't exist.

Possible Solution:

Re-configure the UA Endpoint configuration and reinitialize the server.

The UA Server failed to unregister from the UA Discovery Server. | Endpoint URL: '<endpoint url>'.

Error Type:

Warning

Possible Cause:

1. The UA server endpoint URL and the security policy are not supported in the UA Discovery Server.
2. The attempt to unregister the UA Server from the UA Discovery Server could not complete in the expected manner.

Possible Solution:

Verify the UA Server endpoint URL and the security policy with the UA Discovery Server endpoints.

The UA Server failed to initialize an endpoint configuration. | Endpoint Name: '<name>'.

Error Type:

Warning

Possible Cause:

The endpoint is configured to use a network adapter that does not have a valid ipv4 address.

Possible Solution:

1. Re-configure the network adapter property with an adapter that has a valid ipv4 address.
2. Restart the runtime to refresh the endpoint configurations.

The UA Server successfully registered with the UA Discovery Server. | Endpoint URL: '<endpoint url>'.

Error Type:

Informational

The UA Server successfully unregistered from the UA Discovery Server. | Endpoint URL: '<endpoint url>'.

Error Type:

Informational

Com port is in use by another application. | Port = '<port>'.

Error Type:

Error

Possible Cause:

The serial port assigned to a device is being used by another application.

Possible Solution:

1. Verify that the correct port has been assigned to the channel.
2. Verify that only one copy of the current project is running.

Unable to configure com port with specified parameters. | Port = COM<number>, OS error = <error>.

Error Type:

Error

Possible Cause:

The serial parameters for the specified COM port are not valid.

Possible Solution:

Verify the serial parameters and make any necessary changes.

Driver failed to initialize.

Error Type:

Error

Unable to create serial I/O thread.

Error Type:

Error

Possible Cause:

The server process has no resources available to create new threads.

Possible Solution:

Each tag group consumes a thread. The typical limit for a single process is about 2000 threads. Reduce the number of tag groups in the project.

Com port does not exist. | Port = '<port>'.

Error Type:

Error

Possible Cause:

The specified COM port is not present on the target computer.

Possible Solution:

Verify that the proper COM port is selected.

Error opening com port. | Port = '<port>', OS error = <error>.

Error Type:

Error

Possible Cause:

The specified COM port could not be opened due an internal hardware or software problem on the target computer.

Possible Solution:

Verify that the COM port is functional and may be accessed by other applications.

Connection failed. Unable to bind to adapter. | Adapter = '<name>'.

Error Type:

Error

Possible Cause:

Since the specified network adapter cannot be located in the system device list, it cannot be bound to for communications. This can occur when a project is moved from one PC to another (and when the project specifies a network adapter rather than using the default). The server reverts to the default adapter.

Possible Solution:

Change the Network Adapter property to Default (or select a new adapter), save the project, and retry.

Winsock shut down failed. | OS error = <error>.

Error Type:

Error

Winsock initialization failed. | OS error = <error>.

Error Type:

Error

Possible Solution:

1. The underlying network subsystem is not ready for network communication. Wait a few seconds and restart the driver.
2. The limit on the number of tasks supported by the Windows Sockets implementation has been reached. Close one or more applications that may be using Winsock and restart the driver.

Winsock V1.1 or higher must be installed to use this driver.

Error Type:

Error

Possible Cause:

The version number of the Winsock DLL found on the system is older than 1.1.

Possible Solution:

Upgrade Winsock to version 1.1 or higher.

Socket error occurred binding to local port. | Error = <error>, Details = '<information>'.

Error Type:

Error

Device is not responding.

Error Type:

Warning

Possible Cause:

1. The connection between the device and the host PC is broken.
2. The communication parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect device ID.

4. The response from the device took longer to receive than allowed by the Request Timeout device setting.

Possible Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communications parameters match those of the device.
3. Verify that the device ID for the named device matches that of the actual device.
4. Increase the Request Timeout setting to allow the entire response to be handled.

Device is not responding. | ID = '<device>'.

Error Type:

Warning

Possible Cause:

1. The network connection between the device and the host PC is broken.
2. The communication parameters configured for the device and driver do not match.
3. The response from the device took longer to receive than allowed by the Request Timeout device setting.

Possible Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communications parameters match those of the device.
3. Increase the Request Timeout setting to allow the entire response to be handled.

Serial communications error on channel. | Error mask = <mask>.

Error Type:

Warning

Possible Cause:

1. The serial connection between the device and the host PC is broken.
2. The communications parameters for the serial connection are incorrect.

Possible Solution:

1. Investigate the error mask code and the related information.
2. Verify the cabling between the PC and the PLC device.
3. Verify that the specified communication parameters match those of the device.

See Also:

Error Mask Codes

Invalid array size detected writing to tag <device name>.<address>.

Error Type:

Warning

Possible Cause:

Client trying to write before being updated.

Possible Solution:

Perform a read on the array before attempting a write.

Unable to write to address on device. | Address = '<address>'.

Error Type:

Warning

Possible Cause:

1. The connection between the device and the host PC is broken.
2. The communications parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect device ID.

Possible Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communication parameters match those of the device.
3. Verify that the device ID given to the named device matches that of the actual device.

Items on this page may not be changed while the driver is processing tags.

Error Type:

Warning

Possible Cause:

An attempt was made to change a channel or device configuration while data clients were connected to the server and receiving data from the channel/device.

Possible Solution:

Disconnect all data clients from the server before making changes.

Specified address is not valid on device. | Invalid address = '<address>'.

Error Type:

Warning

Possible Cause:

A tag address has been assigned an invalid address.

Possible Solution:

Modify the requested address in the client application.

Address '<address>' is not valid on device '<name>'.

Error Type:

Warning

This property may not be changed while the driver is processing tags.

Error Type:

Warning

Unable to write to address '<address>' on device '<name>'.

Error Type:

Warning

Possible Cause:

1. The connection between the device and the host PC is broken.
2. The communications parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect device ID.

Possible Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communication parameters match those of the device.
3. Verify that the device ID given to the named device matches that of the actual device.

Socket error occurred connecting. | Error = <error>, Details = '<information>'.

Error Type:

Warning

Possible Cause:

Communication with the device failed during the specified socket operation.

Possible Solution:

Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

Socket error occurred receiving data. | Error = <error>, Details = '<information>'.

Error Type:

Warning

Possible Cause:

Communication with the device failed during the specified socket operation.

Possible Solution:

Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

Socket error occurred sending data. | Error = <error>, Details = '<information>'.

Error Type:

Warning

Possible Cause:

Communication with the device failed during the specified socket operation.

Possible Solution:

Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

Socket error occurred checking for readability. | Error = <error>, Details = '<information>'.

Error Type:

Warning

Possible Cause:

Communication with the device failed during the specified socket operation.

Possible Solution:

Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

Socket error occurred checking for writability. | Error = <error>, Details = '<information>'.

Error Type:

Warning

Possible Cause:

Communication with the device failed during the specified socket operation.

Possible Solution:

Follow the guidance in the error and details, which explain why the error occurred and suggest a remedy when appropriate.

%s |

Error Type:

Informational

<Name> Device Driver '<name>'

Error Type:

Informational

Could not load item state data. Reason: <reason>.

Error Type:

Warning

Possible Cause:

1. The driver could not load the item state data for the specified reason.
2. Corrupt data files.
3. Inadequate disk space.
4. Invalid drive in path.
5. Deleted or renamed data files.

Possible Solution:

Solution depends upon the reason given in the error message. In the case of file corruption or deletion, previous state data is lost.

Could not save item state data. Reason: <reason>.

Error Type:

Warning

Possible Cause:

1. The driver could not save the item state data for the specified reason.
2. Corrupt data files.
3. Inadequate disk space.
4. Invalid drive in path.
5. Deleted or renamed data files.

Possible Solution:

Solution depends upon the reason given in the error message. In the case of file corruption or deletion, previous state data is lost.

Feature '<name>' is not licensed and cannot be used.

Error Type:

Error

Possible Cause:

1. The named feature of the product has not been purchased and licensed.
2. The product license has been removed or trusted storage has become corrupted.

Possible Solution:

1. Download or install the software feature and purchase license.
2. Consult the Licensing User Manual for instructions on activating emergency licenses.
3. Contact a sales or support representative for assistance.

● See Also:

License Utility Help

Failed to load the license interface, possibly due to a missing third-party dependency. Run in Time Limited mode only.

Error Type:

Error

Possible Cause:

One or more required OEM licensing component is missing the system.

Possible Solution:

Contact a sales or support representative for assistance.

● See Also:

License Utility Help

Failed to initialize licensing. Unable to load the demo file license (Error %1!x!).

Error Type:

Error

Possible Cause:

The demo license file is not valid

Failed to initialize licensing. Unable to initialize the licensing identity (Error %1!x!).

Error Type:

Error

Failed to initialize licensing. Unable to initialize trusted storage (Error %1!x!).

Error Type:

Error

Possible Cause:

1. The system identifier has changed
2. Trusted storage has been tampered with

Failed to initialize licensing. Unable to initialize the licensing publisher (Error %1!x!).

Error Type:

Error

Failed to initialize licensing. Unable to establish system time interface (Error %1!x!).

Error Type:

Error

Failed to initialize licensing (Error <error code>)

Error Type:

Error

Failed to process the activation response from the license server (Error: %x, Process Codes: %s, Message Codes: %s)

Error Type:

Error

Failed to create an activation request (Error %x)

Error Type:

Error

Request failed with license server.

Error Type:

Error

Time Limited mode has expired.

Error Type:

Warning

Possible Cause:

1. The product has not been purchased and licensed during Time Limited mode.
2. The server started in Time Limited mode with the specified time remaining in Time Limited mode.

Possible Solution:

1. If evaluating the server, no action needs to be taken.
2. If this is a production machine, activate the product licenses for the installed components before Time Limited mode expires.
3. Purchase a license for all features of the product that will be used.
4. Contact a sales or support representative for assistance.

See Also:

License Utility Help

Maximum device count exceeded for the lite version '<number>' license. Edit project and restart the server.

Error Type:

Warning

Possible Cause:

The specified driver was activated with a lite license, which limits the number of devices that can be configured.

Possible Solution:

1. Verify the number of devices authorized by the license and correct the project design to reduce the device count.
2. If more devices are needed or the lite activation is incorrect, contact a sales representative about upgrading the license to support more devices.

See Also:

License Utility Help

Maximum runtime tag count exceeded for the lite version '<number>' license. Edit client project and restart the server.

Error Type:

Warning

Possible Cause:

The specified driver was activated with a lite license, which limits the number of tags that can be configured.

Possible Solution:

1. Verify the number of tags authorized by the license and correct the project design to reduce the tag count.
2. If more tags are needed or if the lite activation is incorrect, contact a sales representative about upgrading the license to support more tags.

See Also:
License Utility Help

Type <numeric type ID> limit of <maximum count> exceeded on feature '<name>'.

Error Type:
Warning

Possible Cause:
The installed feature license limits the number of items of the specified type that can be configured.

Possible Solution:

1. Contact customer solutions to determine what object type count should be reduced to remain within the limits of the license.
2. If more items are needed, contact a sales representative about upgrading the license.

See Also:
License Utility Help

<Object type name> limit of <maximum count> exceeded on feature '<name>'.

Error Type:
Warning

Possible Cause:
The installed feature license limits the number of items of the specified type that can be configured.

Possible Solution:

1. Verify the number authorized by the license and correct the project design to use only that number of items.
2. If more items are needed, contact a sales representative about upgrading the license.

See Also:
License Utility Help

The FlexNet Licensing Service must be enabled to process licenses. Failure to enable the service results in Time Limited mode.

Error Type:

Warning

Possible Cause:

An attempt was made to verify the license, but the FlexNet Licensing Service is disabled.

Possible Solution:

Use the Windows Service Control Manager to enable the FlexNet Licensing Service, which requires a runtime restart.

See Also:

License Utility Help

The <name> feature license has been removed. The server will enter Time Limited mode unless the license is restored before the grace period expires.

Error Type:

Warning

Possible Cause:

The feature license has been deleted, moved to another machine, the hardware key has been removed, or trusted storage has been corrupted.

Possible Solution:

1. Consult the Licensing User Manual for instructions on activating an emergency licenses.
2. Contact a sales or support representative for assistance.

See Also:

License Utility Help

License for feature <name> cannot be accessed [error=<code>] and must be reactivated.

Error Type:

Warning

Possible Cause:

Trusted storage has been corrupted, possibly due to a system update.

Possible Solution:

1. Consult the Licensing User Manual for instructions on activating an emergency licenses.
2. Contact a sales or support representative for assistance.

See Also:

License Utility Help

Feature %1 is time limited and will expire at %2.

Error Type:

Warning

Feature %1 is time limited and will expire at %2.

Error Type:

Warning

Object count limit has been exceeded on feature <name>. Time limited usage will expire at <date/time>.

Error Type:

Warning

Feature count limit exceeded on <name>. Time limited usage will expire at <date/time>.

Error Type:

Warning

Time limited usage period on feature <name> has expired.

Error Type:

Warning

Failed to obtain licenses from the license server.

Error Type:

Warning

The license for this product has expired and will soon stop functioning. Please contact your sales representative to renew the subscription.

Error Type:

Warning

Licensing for this system is currently provided by a file-based license.

Error Type:

Warning

Failed to connect to the license server.

Error Type:

Warning

Possible Cause:

1. The license server connection parameters are incorrect.
2. The license server is not running or has been disabled.

3. The TLS connection has not been properly configured.

Possible Solution:

1. Verify that the license server connection parameters are correct.
2. Check that the license server is running and that its state is not set to 'suspended'.
3. Verify that the license server CA certificate has been imported.

Failed to return licenses to the LLS.

Error Type:

Warning

Maximum driver count exceeded for the lite version '<name>' driver license. Edit project and restart the server.

Error Type:

Informational

Possible Cause:

The specified driver was activated with a lite license, which limits the number of drivers that can be configured.

Possible Solution:

1. Verify the number of drivers authorized by the license. Correct the project to use only that number of drivers.
2. If more drivers are needed or the lite activation is incorrect, contact a sales representative about upgrading the license to support more drivers.

● See Also:

1. Event Log (in server help)
2. License Utility Help

Connecting to the license server.

Error Type:

Informational

Successful communication with the license server. Renew interval established at %d seconds.

Error Type:

Informational

License synchronization required. Initiating request...

Error Type:

Informational

Performing initial license request to the license server.

Error Type:

Informational

Connected to license server, no changes.

Error Type:

Informational

Requesting return of all borrowed licenses...

Error Type:

Informational

Cannot add item. Requested count of <number> would exceed license limit of <maximum count>.

Error Type:

Informational

Possible Cause:

The product license limits the number of items that can be configured.

Possible Solution:

1. Verify the number authorized by the license and correct the project to use only that number of items.
2. If more items are needed, contact a sales representative about upgrading the license.

● See Also:

License Utility Help

The version of component <name> (<version>) is required to match that of component <name> (<version>).

Error Type:

Informational

Possible Cause:

Two installed components have an interdependency that requires the versions to match.

Possible Solution:

Verify component versions and download or install the matching versions of the components.

● See Also:

License Utility Help

Maximum channel count exceeded for the lite version '<name>' driver license. Edit project and restart the server.

Error Type:

Informational

Possible Cause:

The specified driver was activated with a lite license, which limits the number of channels that can be configured.

Possible Solution:

1. Verify the number of channels authorized by the license. Correct the project to use only that number of channels.
2. If more channels are needed or the lite activation is incorrect, contact a sales representative about upgrading the license to support more channels.

● See Also:

1. Event Log (in server help)
2. License Utility Help

%s is now licensed.

Error Type:

Informational

Appendix — Running ThingWorx Kepware Edge in a Container

ThingWorx Kepware Edge is designed with the ability to run within a container environment. A Docker image built with ThingWorx Kepware Edge and its prerequisites is available to be deployed and run using a variety of tools.

Starting a ThingWorx Kepware Edge Container Instance

To run an instance of the ThingWorx Kepware Edge Docker image, execute the following command:

```
docker run -d -e USE_SAMPLE_PROJECT='<Use sample project flag>' -p 57513:57513 -p
49330:49330 --init --name <Container name> --mount type=bind,source=<Admin password
source directory>,target=/opt/tkedge/v1/secrets --mount type=bind,source=<user data
source directory>, target=/opt/tkedge/v1/user_data --mount type=bind,source=<.config
source directory>, target=/opt/tkedge/v1/.config <Image name>
```

where:

- <Use sample project flag> (optional). Set to TRUE to start with a sample project.
- <Admin password source directory> is on the host machine that contains the password.txt file.
  See the "Administrator Password" on the facing page section for more information.
- <Container name> is the name of the container instance.
- <user data source directory> is on the host machine that is used to share files with the container.
  See the Sharing Files with the Container for more information.
- <.config source directory> is on the host machine that is used to persist the configuration files from the container.
  See the Persisting Data to the Host for more information.
- <Container name> is the name of the container instance.
- <Image name> is the name of the container image.

For additional details about Docker run command options, see Docker Run documentation.

Required Port Binding

The -p option in the Docker run command specifies the port or range of ports to publish from container to host using the format:

```
<Host Port>:<Container Port>
```

The ports in the example above are configured by default in ThingWorx Kepware Edge:

- The default https port for the Configuration API is 57513.
- The default port for UA Endpoints is 49330.

If a port other than the defaults listed above is required, include additional ports when executing the Docker run command. Alternatively, a range of ports can be specified.

*See the **Docker Links User Guide** for information on how to manipulate ports in Docker.*

*See the **Docker Networking Overview** for information on all networking options.*

Additional options can be included in the Docker run command to enable data sharing between the host and container.

*See **Sharing Project Files with the Container** section for an example.*

Administrator Password

A password for the administrator account must be set at container run time. During the container initialization, ThingWorx Kepware Edge searches for a *password.txt* file that contains the administrator account password. The password must be between 14 and 512 characters. Set the permissions on this file such that the Docker container user has read and write permissions. Place this file in a directory accessible to the container via a bind mount, as described in the **Starting a ThingWorx Kepware Edge container** instance section.

Note: When the ThingWorx Kepware Edge startup script is run, it deletes the *password.txt* file.

Caution: An insecure and non-production option is to pass the password as an environmental variable named “EDGEADMINPW” when deploying the container. The password still needs to follow all character and length requirements.

Checking if the Container is Running

View the container status with the command:

```
docker ps -a
```

Sharing Files with the Container

Various configuration and files are necessary to share with the ThingWorx Kepware Edge instance running in the container. For example, project files can be loaded using the Configuration API *projectLoad* service. This service requires files to be located in a specific directory created at container run time:

```
/opt/tkedge/v1/user_data
```

To move any files to the container, a method of file sharing between the host and container must be implemented. The simplest option to share data with the container is to directly copy files into the container file system using the docker cp command:

```
docker cp <source file> <container name>:/opt/tkedge/v1/user_data
```

Persisting Data to the Host

A bind mount can be used to share and persist data used by ThingWorx Kepware Edge with the host machine. This can be accomplished by adding the `--mount` option to the Docker run command:

```
docker run -d -p 57513:57513 -p 49330:49330 --init --name <Container name> --mount
type=bind,source=<user data source directory>,target=/opt/tkedge/v1/user_data --mount
type=bind,source=<.config source directory>,target=/opt/tkedge/v1/.config <Image
name>
```

Note: The target parameters must not be modified from this example. These directories are created at container run time for the purpose of storing application data and are not configurable.

Persisting configuration data is strongly recommended while using ThingWorx Kepware Edge in a container. Configuration data, such as the project file, OPC UA certificates and endpoints, user management configuration and other data are stored in the */opt/tkedge/v1/.config* folder in the containers file system. Persisting this folder allows for a container to be redeployed due to failure or planned updates while keeping all configuration data from the previous running state.

To persist the configuration data add a volume mount mapped to the *.config* directory as seen in the Docker run command above.

Note: Do not mount more than one container to a shared *.config* directory. Each unique instance of ThingWorx Kepware Edge needs its own data store for configuration. This is not supported and can result in undefined behavior.

See ***Backup and Restore*** *configuration data.*

*For additional details about sharing data between containers using Docker volume mounts., see **Docker Volume documentation**.*

Permissions

To access the specified data source directory on the host, user and group entities identical to those created for ThingWorx Kepware Edge at container run time must exist on the host and be granted the appropriate permission on that directory, where group: tkedge and user: tkedge.

Note: The host and container user and group entities must have matching UID and GID.

Configuring a ThingWorx Kepware Edge Container Instance:

The ThingWorx Kepware Edge instance is operational after executing the above "docker run" command. To manage certificates for northbound interfaces or configure other administrative options, connect to a command shell on the container with the following command:

```
docker exec -it <Container Name> /bin/bash
```

From the command shell, the "edge_admin" command-line tool can be used to perform these actions.

*See the **Command-Line Edge Admin** for more information on this tool.*

Note: The ThingWorx Kepware Edge runtime must be reinitialized through the Configuration API or restarted after making changes to the UA Endpoint configuration. Restarting the container can be accomplished by running the "docker stop" followed by "docker start" command.

Managing OPC UA Certificates

The preferred method for managing OPC UA certificates is to share the trusted certificates through the mounted .config folder. This allows trusted client certificates to be added or updated in the trust store without connecting to the container with a command shell.

*See **Managing OPC UA Certificates** through the .config folder.*

Event Log

In a container environment, log services, such as Docker log service, are used to monitor information about the running container. ThingWorx Kepware Edge can be configured to send all event log messages to * STDOUT* to make the messages accessible through the docker log service.
To enable this, use the Configuration API and set the Log to Console properties in the Admin properties as shown below.

Endpoint (PUT):

```
https://<hostname_or_ip>:<port>/config/v1/admin
```

Body:

```
{ "libadminsettings.EVENT_LOG_LOG_TO_CONSOLE": true }
```

*For additional details about monitoring the Docker log service, see the **Docker log documentation**.*

Appendix — Running ThingWorx Kepware Edge as an Azure IoT Edge Module

ThingWorx Kepware Edge is available to deploy within an Azure IoT environment with access to the module provided in the **Azure Marketplace**. This allows for the ability to manage and deploy ThingWorx Kepware Edge within an environment that is using the Azure IoT Edge runtime.

Prerequisites

- Azure IoT Portal Account
- Azure IoT Hub Deployed (*see* ***Instructions***)
- IoT Edge runtime environment must be installed on the host
- IoT Edge runtime needs to be installed and deployed to an IoT Edge device

See Also*: ThingWorx Kepware Edge module* ***installation instructions****.*

Managing Configuration

An IoT Edge module is functionally a container used to deploy an application. Managing the configuration of the module for Azure IoT is like managing ThingWorx Kepware Edge in a container. Persisting configuration data to the host IoT device environment is necessary to ensure that application data is accessible during various module management actions such as deployment, upgrade, etc.

In preparation for a production deployment, three folders should be configured for access from the module to support persisting the configuration data (.config folder), user data (user_data folder) and the secrets data (secrets folder).

For additional details about these application folders for ThingWorx Kepware Edge, see sections ***Sharing Files with the Container and Persisting Data to the Host****.*

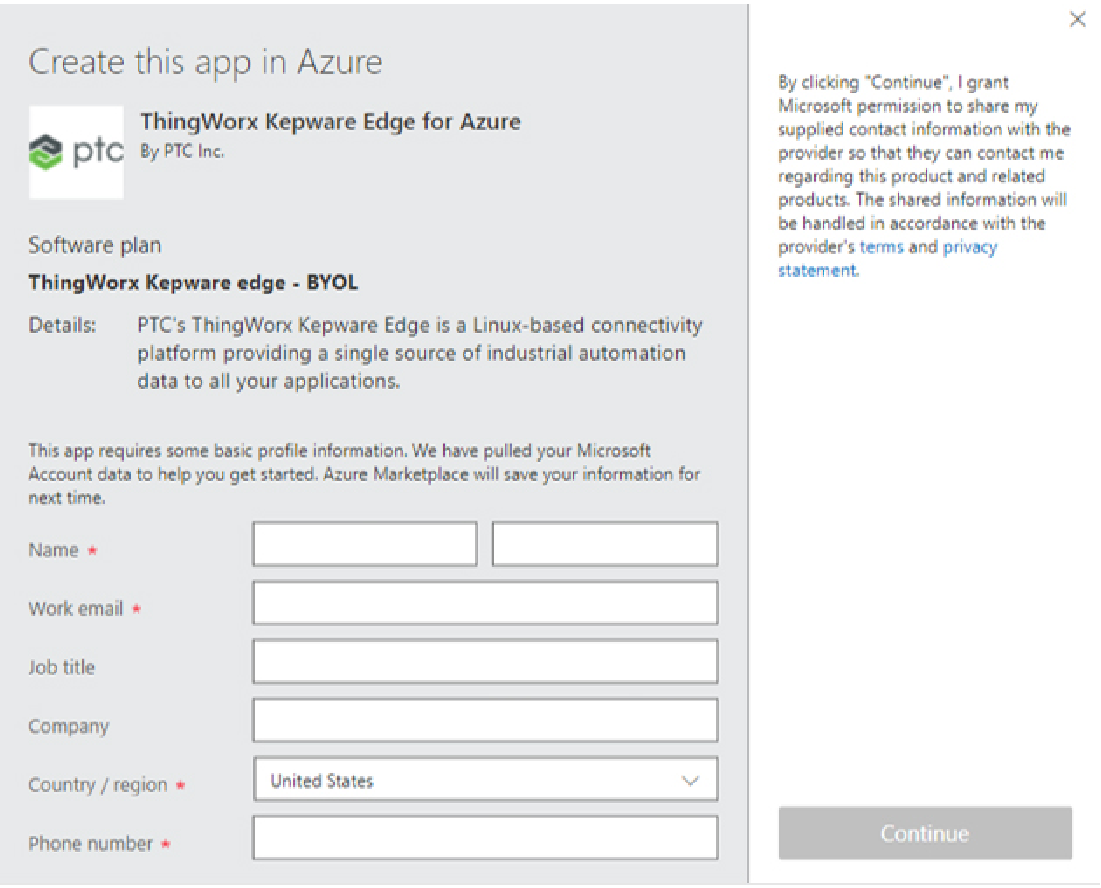
For additional details about sharing data between modules and the IoT Edge runtime host, see Azure's ***"How to Access Host Storage from Module"*** *documentation.*

Deploying ThingWorx Kepware Edge Module

A IoT Edge module can be deployed using multiple tools within the Azure ecosystem. This section uses the Azure Portal and the Azure Marketplace to select and deploy the ThingWorx Kepware Edge for Azure module.

1. Open the Azure Marketplace listing for the ThingWorx Kepware Edge for Azure module.
2. Select **Get It Now** button, fill out form, and accept the BOYL agreement.
3. Sign into a valid Azure account if not already signed in.

   Confirmation of contact information is necessary as part of the initial access to the module on the Azure marketplace.



Create this app in Azure

ThingWorx Kepware Edge for Azure

By PTC Inc.

Software plan

ThingWorx Kepware edge - BYOL

Details: PTC's ThingWorx Kepware Edge is a Linux-based connectivity platform providing a single source of industrial automation data to all your applications.

This app requires some basic profile information. We have pulled your Microsoft Account data to help you get started. Azure Marketplace will save your information for next time.

Name *

Work email *

Job title

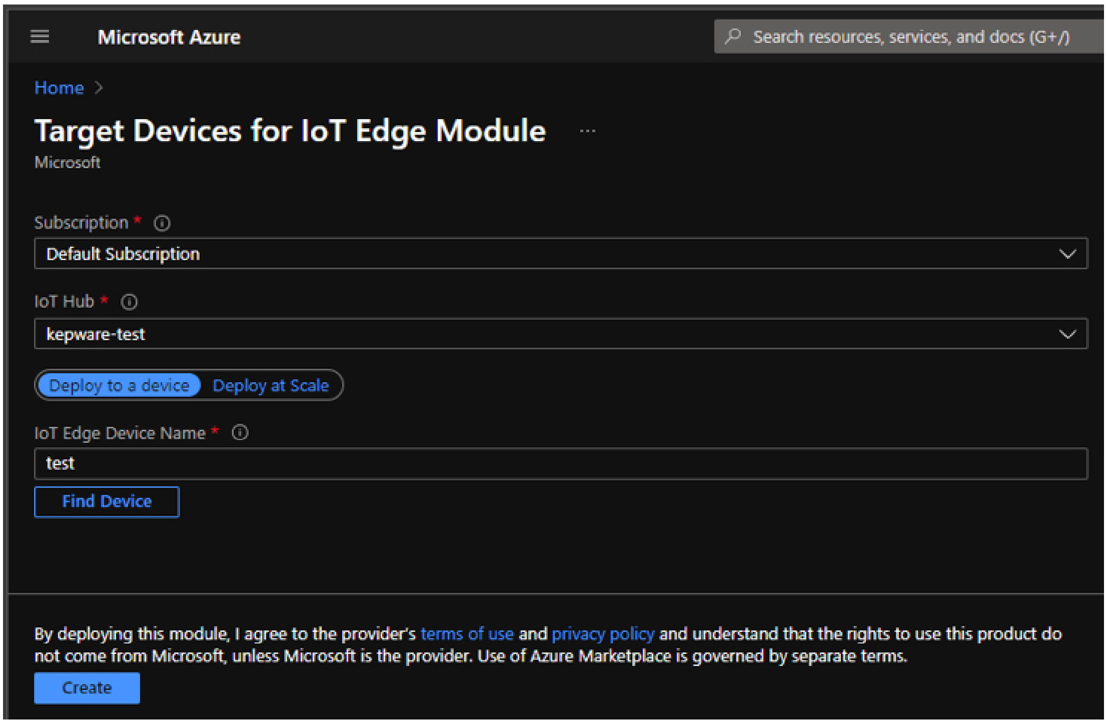
Company

Country / region * United States

Phone number *

By clicking "Continue", I grant Microsoft permission to share my supplied contact information with the provider so that they can contact me regarding this product and related products. The shared information will be handled in accordance with the provider's terms and privacy statement.

4. An Azure portal page opens in a browser to configure the target IoT Edge device to receive the module. Select the appropriate Subscription, IoT Hub, and IoT Edge Device to deploy the module to and select **Create**.



Target Devices for IoT Edge Module

Microsoft

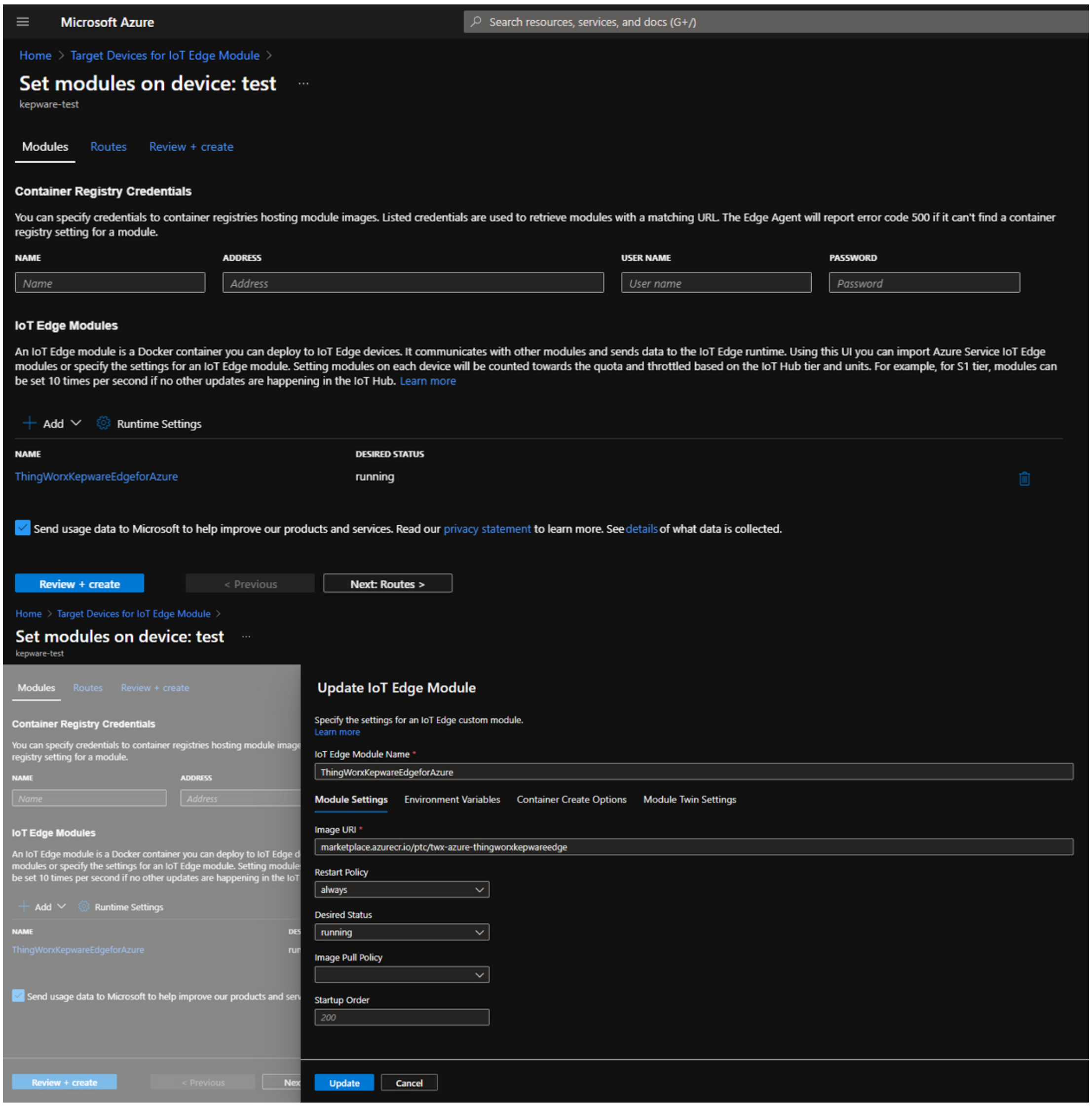
Subscription *: Default Subscription

IoT Hub *: kepware-test

IoT Edge Device Name *: test

By deploying this module, I agree to the provider's terms of use and privacy policy and understand that the rights to use this product do not come from Microsoft, unless Microsoft is the provider. Use of Azure Marketplace is governed by separate terms.

5. Once a target IoT Edge device has been selected, confirm and/or modify the Module settings, Environment Variables, and Container Create Options.
   ● **Caution**: Do NOT click **Review and Create** until Step 10.



Microsoft Azure

Search resources, services, and docs (G+/)

Home > Target Devices for IoT Edge Module >

Set modules on device: test

kepware-test

Modules Routes Review + create

Container Registry Credentials

You can specify credentials to container registries hosting module images. Listed credentials are used to retrieve modules with a matching URL. The Edge Agent will report error code 500 if it can't find a container registry setting for a module.

NAME	ADDRESS	USER NAME	PASSWORD
Name	Address	User name	Password

IoT Edge Modules

An IoT Edge module is a Docker container you can deploy to IoT Edge devices. It communicates with other modules and sends data to the IoT Edge runtime. Using this UI you can import Azure Service IoT Edge modules or specify the settings for an IoT Edge module. Setting modules on each device will be counted towards the quota and throttled based on the IoT Hub tier and units. For example, for S1 tier, modules can be set 10 times per second if no other updates are happening in the IoT Hub. Learn more

+ Add Runtime Settings

NAME	DESIRED STATUS
ThingWorxKepwareEdgeforAzure	running

☑ Send usage data to Microsoft to help improve our products and services. Read our privacy statement to learn more. See details of what data is collected.

Review + create < Previous Next: Routes >

Home > Target Devices for IoT Edge Module >

Set modules on device: test

kepware-test

Update IoT Edge Module

Specify the settings for an IoT Edge custom module.
Learn more

IoT Edge Module Name *
ThingWorxKepwareEdgeforAzure

Module Settings Environment Variables Container Create Options Module Twin Settings

Image URI *
marketplace.azurecr.io/ptc/twx-azure-thingworxkepwareedge

Restart Policy
always

Desired Status
running

Image Pull Policy

Startup Order
200

Update Cancel

6. Select the **ThingworxKepwareEdgeforAzure** module to view the module options.

7. Select the **Container Create Options** tab and modify the options as necessary. Below is an example of the options that could be used.

```
{
    "ExposedPorts": {
       "57513/tcp": {},
       "49330/tcp": {}
    },
    "HostConfig": {
       "PortBindings": {
             "57513/tcp": [
                {
                   "HostPort": "57513"
                }
             ],
             "49330/tcp": [
                {
                "HostPort": "49330"
                }
             ]
      },
      "Binds": [
      "<HostStoragePath>/config:/opt/tkedge/v1/.config",
      "<HostStoragePath>/user_data:/opt/tkedge/v1/user_data",
      "<HostStoragePath>/secrets:/opt/tkedge/v1/secrets"
      ]
    }
}
```

“Binds” configurations are used for sharing data to the module from the IoT Edge Device host environment and is used to persist configuration data, share files, and securely assign the Administrator password for ThingWorx Kepware Edge. Modify the <HostStoragePath> to point to the appropriate folder locations on the IoT Edge device host used to persist files from the module.

Ensure that the appropriate permissions are set to the <HostStoragePath> directories.

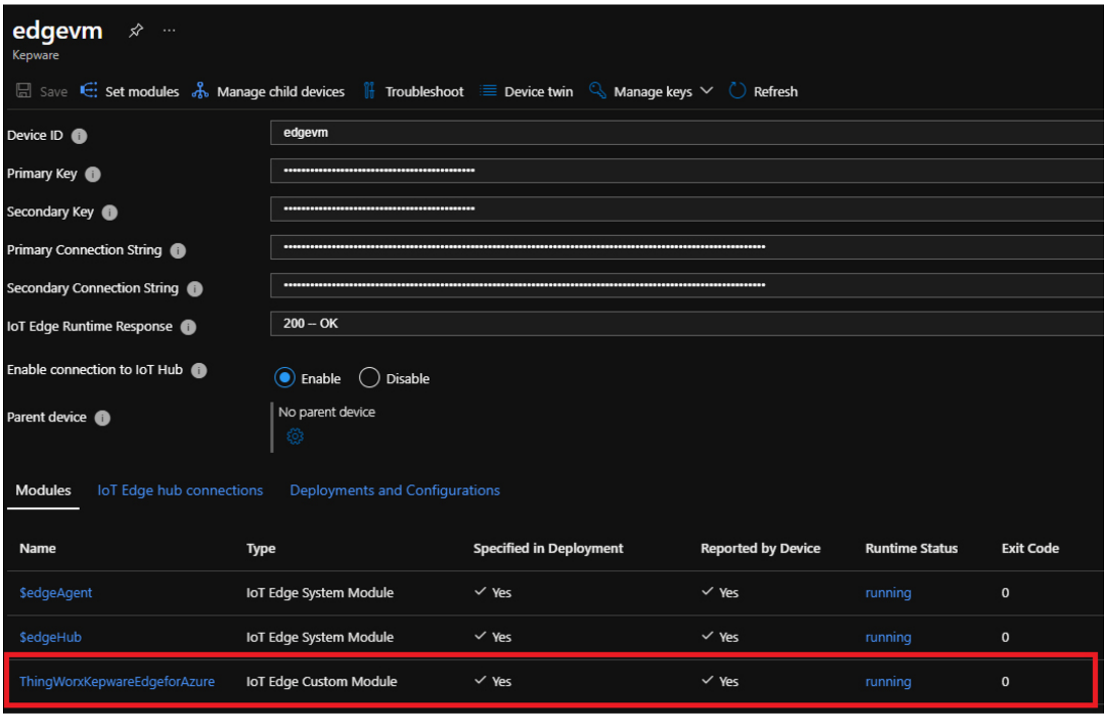
*For additional details about these application folders and permissions, see sections **Sharing Files with the Container**, **Persisting Data to the Host**, and **Permissions**.*

8. Select the **Environment Variables** tab.

   For secure deployments, it is recommended to delete the EDGEADMINPW and use the “secrets” bind to initialize the container with an Administrator password. For insecure non-production testing purposes, it is possible to set the EDGEADMINPW environmental variable value to assign a password at deployment.

   *For additional details about Administrator password assignment, see **Administrator Password**.*

9. Once completed, select **Update**.
10. Select **Review** then **Create** buttons to deploy the module.
11. From the IoT Edge device information interface, confirm that the module is listed and is in the running state.



Device ID: edgevm

Primary Key: ••

Secondary Key: ••

Primary Connection String: ••

Secondary Connection String: ••

IoT Edge Runtime Response: 200 – OK

Enable connection to IoT Hub: ● Enable ○ Disable

Parent device: No parent device

Name	Type	Specified in Deployment	Reported by Device	Runtime Status	Exit Code
$edgeAgent	IoT Edge System Module	✓ Yes	✓ Yes	running	0
$edgeHub	IoT Edge System Module	✓ Yes	✓ Yes	running	0
ThingWorxKepwareEdgeforAzure	IoT Edge Custom Module	✓ Yes	✓ Yes	running	0

Index

%

%s | 126

%s is now licensed. 135

<

<feature name> is required to load this project. 107

<feature name> was not found or could not be loaded. 107

<Name> Device Driver '<name>' 126

<Object type name> limit of <maximum count> exceeded on feature '<name>'. 130

<Product> device driver loaded successfully. 112

<Product> device driver unloaded from memory. 113

A

A client application has disabled auto-demotion on device '<device>'. 114

A client application has enabled auto-demotion on device '<device>'. 113

A socket error occurred listening for client connections. | Endpoint URL = '<endpoint URL>', Error = <error code>, Details = '<description>'. 117

Access to object denied. | User = '<account>', Object = '<object path>', Permission = 115

Address '<address>' is not valid on device '<name>'. 124

Administrator 39

Alias Name 36

Alias Properties 36

Anonymous 77

APPKEY 47

Application Data 12

Architecture 11, 48, 67

Attempting to automatically generate tags for device '<device>'. 113

authentication 52

Authentication 39, 102

Authorization 39

Auto-generated tag '<tag>' already exists and will not be overwritten. 108

Auto generation produced too many overwrites, stopped posting error messages. 109

Automatic Tag Generation 68

Azure 139

B

Basic256Sha256 103

BCD 33

Boolean 33

Byte 33

C

cacerts 50

Cannot add device. A duplicate device may already exist in this channel. 108

Cannot add item. Requested count of <number> would exceed license limit of <maximum count>. 134

certificate 50

Certificates 15

Char 33

Child Endpoints 90

Clamp 35

Com port does not exist. | Port = '<port>'. 120

Com port is in use by another application. | Port = '<port>'. 119

Command line 10

Command Line Interface 15

Command line interfaces 10

Completed automatic tag generation for device '<device>'. 113

Components and Concepts 16

Concurrent Clients 52

config_api_service 40

Configuration API Service 52

Configuration API Service — Configuring Licensing Server 95

Configuration API started with SSL on port <port number>. 105

Configuration API started without SSL on port <port number>. 104

Configuration Backup and Restore 40

Configuring User Group Project Permissions 95

Connected to license server, no changes. 134

Connecting to the license server. 133

Connecting with an OPC UA Client with UaExpert 102

Connection failed. Unable to bind to adapter. | Adapter = '<name>'. 120

Connectivity 41, 45

Content Retrieval 54

Could not load item state data. Reason <reason>. 126

Could not save item state data. Reason <reason>. 126

Create MQTT Agent 49

Create MQTT Agent Tag 50

Created backup of project '<name>' to '<path>'. 114

Creating a Channel 79

Creating a Device 82

Creating a Tag 84

Creating a UA Endpoint 100

Creating a User 93

Creating a User Group 93

Credentials 77

cURL 10

Curl 79

D

Data 63

Data collection is disabled on device '<device>'. 113

Data collection is enabled on device '<device>'. 113

Default 12

DELETE 81, 84, 86, 88

Delete MQTT Agent 50

Demo License 13

Device '<device>' has been auto-promoted to determine if communications can be re-established. 114

Device '<device>' has been automatically demoted. 111

Device discovery has exceeded <count> maximum allowed devices. Limit the discovery range and try again. 107

Device is not responding. 121

Device is not responding. | ID = '<device>'. 122

Directory 12

Documentation Endpoint 41

Documentation Endpoints 42

Double 33

Driver failed to initialize. 120

DWord 33

Dynamic Tags 33

E

Edge Module 139

edge_admin 15

Enable 102

Endpoint 39, 97

Endpoint Mapping 41

Error opening com port. | Port = '<port>', OS error = <error>. 120

Evaluation 13

Event Log Messages 104

F

Failed to add tag '<tag>' because the address is too long. The maximum address length is <number>. 109

Failed to connect to the license server. 132

Failed to create an activation request (Error %x) 128

Failed to import server instance cert
'<cert location>'. Please use the OPC UA Configuration Manager to reissue the certificate. 116

Failed to initialize licensing (Error <error code>) 128

Failed to initialize licensing. Unable to establish system time interface (Error %1!x!). 128

Failed to initialize licensing. Unable to initialize the licensing identity (Error %1!x!). 127

Failed to initialize licensing. Unable to initialize the licensing publisher (Error %1!x!). 128

Failed to initialize licensing. Unable to initialize trusted storage (Error %1!x!). 128

Failed to initialize licensing. Unable to load the demo file license (Error %1!x!). 127

Failed to load library
<name>. 114

Failed to load the license interface, possibly due to a missing third-party dependency. Run in Time Limited mode only. 127

Failed to load the UA Server endpoint configuration. 118

Failed to obtain licenses from the license server. 132

Failed to process the activation response from the license server (Error
%x, Process Codes
%s, Message Codes
%s) 128

Failed to read build manifest resource
<name>. 114

Failed to return licenses to the LLS. 133

Feature '<name>' is not licensed and cannot be used. 127

Feature %1 is time limited and will expire at %2. 132

Feature count limit exceeded on <name>. Time limited usage will expire at <date/time>. 132

Filtering 53, 60

Float 33

FORCE_UPDATE 44

G

GET Request URI 54

Getting Started 38

Group has been deleted. | Group = '<name>'. 116

H

Health Status Endpoint 42

Health Status Endpoints 42

Hierarchy 64

HOSTNAME 47

HTTP 52

HTTPS 52

Human Machine Interface (HMI) 45

I

Initialization 52

Insomnia 79

install 11

Installer 11

installing 11

Instance Certificate 15

Interface 44-45

Interfaces and Connectivity 45

Introduction 10

Invalid array size detected writing to tag <device name>.<address>. 123

Invalid Model encountered while trying to load the project. | Device = '<device>'. 108

Invalid project file. 105

IoT Gateway 49-50

IoT Gateway — MQTT 48

Items on this page may not be changed while the driver is processing tags. 123

J

Java Runtime 10

Job 68

Job Cleanup 68

JSON Response Structure 54

K

KeyStore 50

keytool 50

L

LBCD 33

License for feature <name> cannot be accessed [error=<code>] and must be reactivated. 131

License Recheck 14

License Server 13

License synchronization required. Initiating request... 134

Licensing 13

Licensing for this system is currently provided by a file-based license. 132

Linear 34

Linux 10

LLong 33

Location 12

Log Endpoints 42

Logging 53

Long 33

LSB 10

M

Man Machine Interface (MMI) 45

Mapped to 36

Maximum channel count exceeded for the lite version '<name>' driver license. Edit project and restart the server. 135

Maximum device count exceeded for the lite version '<number>' license. Edit project and restart the server. 129

Maximum driver count exceeded for the lite version '<name>' driver license. Edit project and restart the server. 133

Maximum runtime tag count exceeded for the lite version '<number>' license. Edit client project and restart the server. 129

Member 56

Missing server instance certificate '<cert location>'. Please use the OPC UA Configuration Manager to reissue the certificate. 116

MQTT 48

MQTT Agent 44, 49-50

MQTT client 10

Multiple Objects 63

N

Negate 35

No tags were created by the tag generation request. See the event log for more information. 112

O

Object 63

Object count limit has been exceeded on feature <name>. Time limited usage will expire at <date/time>. 132

Object type '<name>' not allowed in project. 114

OPC UA 45

OPC UA Certificate Management 46

OPC UA Endpoint 97

OPC UA server 102

OpenJDK 10

Operation 52

P

password 39

Password 58, 77

Password for user has been changed. | User = '<name>'. 116

Performing initial license request to the license server. 134

Permissions definition has changed on user group. | Group = '<name>'. 115

Plug-in Endpoints 42

Port 39, 102

PORT 47

Ports 38

Postman 10, 79

Prerequisites 49

Project 39

Project Permissions 90

Project Properties — OPC UA 77

Project Properties (via API Commands) 73

Project Save 70

ProjectSave 72

Properly Name a Channel, Device, Tag, and Tag Group 36

Property Definitions 56

Property Tags 30

Property Types 58

Property Validation Error Object 88

Q

QWord 33

R

Raw 34

Reinitialize Runtime Service 71

Rejecting attempt to change model type on a referenced device '<channel device>'. 110

Removing a Device 84

Removing a Tag 86

Removing a Tag Group 88

Removing a UA Endpoint 100

Removing Channel 81

Request failed with license server. 128

Requesting return of all borrowed licenses... 134

Response Codes 73

REST 39, 52, 79, 82, 84

Running in a Container 136

S

Save 12

Scaled 35

Scan rate override 36

security 12, 39

Security 52, 54, 67, 77, 92, 102

Self-Signed Certificates 50

Serial communications error on channel. | Error mask = <mask>. 122

Server Administration Endpoints 42

server_eventlog 40

server_runtime 40

Service 67

Service Logs 38

Services 38

Short 33

Shutdown 52

Simulation mode is disabled on device '<device>'. 113

Simulation mode is enabled on device '<device>'. 113

Socket error occurred binding to local port. | Error = <error>, Details = '<information>'. 121

Socket error occurred checking for readability. | Error = <error>, Details = '<information>'. 125

Socket error occurred checking for writability. | Error = <error>, Details = '<information>'. 125

Socket error occurred connecting. | Error = <error>, Details = '<information>'. 124

Socket error occurred receiving data. | Error = <error>, Details = '<information>'. 125

Socket error occurred sending data. | Error = <error>, Details = '<information>'. 125

Sorting 60

Specified address is not valid on device. | Invalid address = '<address>'. 123

Square Root 34

Starting <name> device driver. 112

Statistics Tags 31

Stopping <name> device driver. 113

String 33

Successful communication with the license server. Renew interval established at %d seconds. 133

System Requirements 10

System Services 67

System Tags 18

T

Tag count 13

Tag generation results for device '<device>'. | Tags created = <count>, Tags not overwritten = <count>. 114

Tag generation results for device '<device>'. | Tags created = <count>, Tags overwritten = <count>. 114

Tag generation results for device '<device>'. | Tags created = <count>. 114

Tag Group Properties 35

Tag Limit 13

Tag Properties — General 17

Tag Properties — Scaling 34

The <name> device driver was not found or could not be loaded. 105

The <name> feature license has been removed. The server will enter Time Limited mode unless the license is restored before the grace period expires. 131

The Config API is unable to load the SSL certificate. 104

The Config API SSL certificate contains a bad signature. 104

The Config API SSL certificate has expired. 104

The Config API SSL certificate is self-signed. 104

The configured version of TLS for the Configuration API is no longer considered secure. It is recommended that only TLS 1.2 or higher is used. 104

The endpoint '<url>' has been added to the UA Server. 116

The endpoint '<url>' has been disabled. 116

The endpoint '<url>' has been enabled. 116

The endpoint '<url>' has been removed from the UA Server. 116

The FlexNet Licensing Service must be enabled to process licenses. Failure to enable the service results in Time Limited mode. 130

The license for this product has expired and will soon stop functioning. Please contact your sales representative to renew the subscription. 132

The specified network adapter is invalid on channel '%1' | Adapter = '%2'. 112

The UA server certificate is expired. Please use the OPC UA Configuration Manager to reissue the certificate. 117

The UA Server failed to initialize an endpoint configuration. | Endpoint Name '<name>'. 119

The UA Server failed to register with the UA Discovery Server. | Endpoint URL '<endpoint url>'. 117

The UA Server failed to unregister from the UA Discovery Server. | Endpoint URL '<endpoint url>'. 118

The UA Server successfully registered with the UA Discovery Server. | Endpoint URL '<endpoint url>'. 119

The UA Server successfully unregistered from the UA Discovery Server. | Endpoint URL '<endpoint url>'. 119

The version of component <name> (<version>) is required to match that of component <name> (<ver-

sion>). 134

THING_NAME 47

ThingWorx Native Interface 44

ThingWorx Native Interface Certificate Management 48

ThingWorx Native Interface Example 46

ThingWorx Platform 10

This property may not be changed while the driver is processing tags. 124

Time Limited mode has expired. 128

Time limited usage period on feature <name> has expired. 132

Trust Store 15

Type <numeric type ID> limit of <maximum count> exceeded on feature '<name>'. 130

Type Definitions 56

U

UA Server 44

uaserver 103

Ubuntu 10

Unable to add channel due to driver-level failure. 106

Unable to add device due to driver-level failure. 106

Unable to backup project file to '<path>' [<reason>]. The save operation has been aborted. Verify the destination file is not locked and has read/write access. To continue to save this project without a backup, deselect the backup option under Tools | Options | General and re-save the project. 107

Unable to configure com port with specified parameters. | Port = COM<number>, OS error = <error>. 119

Unable to create serial I/O thread. 120

Unable to generate a tag database for device '<device>' 109

Unable to generate a tag database for device '<device>'. The device is not responding. 108

Unable to load driver DLL '<name>'. 110

Unable to load driver DLL '<name>'. Reason 111

Unable to load plug-in DLL '<name>'. 111

Unable to load plug-in DLL '<name>'. Reason 112

Unable to load project <name> 106

Unable to load the '<name>' driver because more than one copy exists ('<name>' and '<name>'). Remove the conflicting driver and restart the application. 105

Unable to load the project due to a missing object. | Object = '<object>'. 108

Unable to save project file <name> 107

Unable to start the Config API Service. Possible problem binding to port. 104

Unable to start the UA server due to certificate load failure. 118

Unable to use network adapter '<adapter>' on channel '<name>'. Using default network adapter. 109

Unable to write to address '<address>' on device '<name>'. 124

Unable to write to address on device. | Address = '<address>'. 123

Update MQTT Agent 50

Updating a Channel 80

Updating a Device 83

Updating a Tag 85

Updating a Tag Group 87

Updating a UA Endpoint 100

Updating a User 94

Updating a User Group 94

URL 102

User added to user group. | User = '<name>', Group = '<name>'. 115

User group has been created. | Group = '<name>'. 115

User group has been disabled. | Group = '<name>'. 115

User group has been enabled. | Group = '<name>'. 116

User group has been renamed. | Old name = '<name>', New name = '<name>'. 115

User Groups 89

User has been deleted. | User = '<name>'. 116

User has been disabled. | User = '<name>'. 115

User has been enabled. | User = '<name>'. 115

User has been renamed. | Old name = '<name>', New name = '<name>'. 115

User Management 88

User moved from user group. | User = '<name>', Old group = '<name>', New group = '<name>'. 115

Users 92

V

Validation error on '<tag>'

- <error>. 110
- Invalid scaling parameters. 110

Version mismatch. 106

View MQTT Agent Tags 50

View MQTT Agents 49

W

What is a Channel? 16

What is a Device? 16

What is a Tag Group? 35

What is a Tag? 17

What is the Alias Map? 36

What is the Event Log? 36

Winsock initialization failed. | OS error = <error>. 121

Winsock shut down failed. | OS error = <error>. 121

Winsock V1.1 or higher must be installed to use this driver. 121

Word 33