thingworx kepware edge

Quick Start Guide

Table of Contents

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.*

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).

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***

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
```

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 **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>
```

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.

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

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 — 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 — Allen-Bradley ControlLogix Ethernet Example

For a list of channel and device definitions and enumerations, access the following endpoints with the REST client or refer to the appendices.

Channel Definitions

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/doc/drivers/Allen-Bradley%20ControlLogix%20Ethernet/channels
```

Device Definitions

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/doc/drivers/Allen-Bradley%20ControlLogix%20Ethernet/devices
```

Create Allen-Bradley ControlLogix Ethernet Channel

Endpoint (POST):

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

Body:

```
{
   "common.ALLTYPES_NAME": "MyChannel",
   "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Allen-Bradley ControlLogix Ethernet"
}
```

Create Allen-Bradley ControlLogix Ethernet Device

Endpoint (POST):

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

Body:

```
{
   "common.ALLTYPES_NAME": "MyDevice",
   "servermain.DEVICE_ID_STRING": "<IP Address>,0,1",
   "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Allen-Bradley ControlLogix Ethernet",
   "servermain.DEVICE_MODEL": <model enumeration>
}
```

where <IP Address> is the device's IP address.

Note: The format of the value for servermain.DEVICE_ID_STRING may vary depending on the model enumeration specified for servermain.DEVICE_MODEL. The device ID string format shown above is for the ControlLogix 5500 model.

Create Allen-Bradley ControlLogix Ethernet Tags

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/MyChannel/devices/MyDevice/tags
```

Body:

```
[
{
   "common.ALLTYPES_NAME": "MyTag1",
   "servermain.TAG_ADDRESS": "My_Tag_Address"
}
{
   "common.ALLTYPES_NAME": "MyTag2",
   "servermain.TAG_ADDRESS": "My_Tag_Address"
}
]
```

See server and driver-specific help for more information on configuring tags and tag groups using the Configuration API.

Configuration API — Modbus TCP/IP Ethernet Example

For a list of channel and device definitions and enumerations, access the following endpoints with the REST client or refer to the appendices.

Channel Definitions

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/config/v1/doc/drivers/Modbus%20TCP%2FIP%20Ethernet/channels
```

Device Definitions

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/doc/drivers/Modbus%20TCP%2FIP%20Ethernet/devices
```

Create Modbus TCP/IP Ethernet Channel

Endpoint (POST):

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

Body:

```
{
   "common.ALLTYPES_NAME": "MyChannel",
    "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet"
}
```

Create Modbus TCP/IP Ethernet Device

Endpoint (POST):

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

Body:

```
{
    "common.ALLTYPES_NAME": "MyDevice",
    "servermain.DEVICE_ID_STRING": "<IP Address>.<Modbus ID>",
    "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet",
    "servermain.DEVICE_MODEL": <model enumeration>
}
```

Device ID Update

Update the Device ID using a “PUT” command from a REST client.
The Endpoint example below references the “demo-project.json” project configuration with “ModbusTCPIP” channel name and “ModbusDevice” device name.

Device ID Example

Endpoint (PUT):

```
https://<hostname_or_ip>:<-
port>/config/v1/project/channels/ModbusTCPIP/devices/ModbusDevice
```

Body:

```
{
    "project_id": <project_ID_from_GET>,
    "servermain.DEVICE_ID_STRING": "<IP Address>.<Modbus ID>"
}
```

Create Modbus TCP/IP Ethernet Tags

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/MyChannel/devices/MyDevice/tags
```

Body:

```
[
    {
        "common.ALLTYPES_NAME": "MyTag1",
        "servermain.TAG_ADDRESS": "40001"
     }
     {
        "common.ALLTYPES_NAME": "MyTag2",
        "servermain.TAG_ADDRESS": "40002"
    }
]
```

See server and driver-specific help for more information on configuring projects over the Configuration API.

Configuration API — Siemens TCP/IP Ethernet Example

For a list of channel and device definitions and enumerations, access the following endpoints with the REST client or refer to the appendices.

Channel Definitions

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/config/v1/doc/drivers/Siemens%20TCP%2FIP%20Ethernet/channels
```

Device Definitions

Endpoint (GET):

```
https://<hostname_or_ip>:<-
port>/config/v1/doc/drivers/Siemens%20TCP%2FIP%20Ethernet/devices
```

Create Siemens TCP/IP Ethernet Channel

Endpoint (POST):

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

Body:

```
{
"common.ALLTYPES_NAME": "MyChannel",
"servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Siemens TCP/IP Ethernet"
}
```

Create Siemens TCP/IP Ethernet Device

Endpoint (POST):

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

Body:

```
{
"common.ALLTYPES_NAME": "MyDevice",
"servermain.DEVICE_ID_STRING": "<IP Address>",
"servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Siemens TCP/IP Ethernet",
"servermain.DEVICE_MODEL": <model enumeration>
}
```

Tip: The above minimum required properties are adequate to define a NetLink device as well.

Create Siemens TCP/IP Ethernet Tags

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/MyChannel/devices/MyDevice/tags
```

Body:

```
[
{
"common.ALLTYPES_NAME": "MyTag1",
"servermain.TAG_ADDRESS": "DB1,INT00"
}
{
"common.ALLTYPES_NAME": "MyTag2",
"servermain.TAG_ADDRESS": "DB1,INT01"
}
]
```

● *See server and driver-specific help for more information on configuring projects over the Configuration API.*

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>"
}
```

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
```

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.

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 — Log Retrieval

Messages from the event log service can be read from a REST client by sending a GET to https://<hostname>:<port>/config/v1/event_log. Messages from the API transaction log service can be read from a REST client by sending a GET to https://<hostname>:<port>/config/v1/transaction_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.*

Event Log

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

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 event type (Information, Warning, Error, Security) or 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?event-
t=Warning,Error&limit=10&start=2016-01-01T00:00:00.000&end=2016-01-02T20:00:00.000
```

where:

1. **event** = Event type to filter. Multiple event types can be provided as comma-separated list. For instance, event=Information,Warning,Error,Security. Selects all event types.

2. **limit** = Maximum number of log entries to return. The default setting is 100 entries.

3. **start** = Earliest time to be returned in YYYY-MM-DDTHH:mm:ss.sss (UTC) format.

4. **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.

Sorting

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/event_log?sortProperty=event&sortOrder=ascending
```

where:

- **sortProperty**: The property to sort by (timestamp, event, source, message)
- **sortOrder**: The sort order (ascending or descending)

Pagination

- **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 is an example of adding the pagination parameters to the eventlog endpoint:

- Requesting both pageSize and pageNumber:

```
https://<hostname_or_ip>:<port>/config/v1/event_log?pageNumber=1&pageSize=10
```

Note: Sorting and pagination of the eventlog is limited to the first 100,000 records. This means in Extended Data Store persistence mode, records beyond 100,000 are not considered for sorting and pagination.