IoT Gateway

Table of Contents

IoT Gateway

Help version 1.024

CONTENTS

Overview

The "Internet of Things" IoT Gateway is an optional feature of KEPServerEX 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 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
- Support for publishing via MQTT, ThingWorx and REST client agents
- Support for reading data from MQTT and REST server agents
- Configurable data collection rate, as frequent as 10 milliseconds up to once per 27.77 hours (99999990 milliseconds) for the REST and MQTT Client
- Configurable data publish rate, as frequent as 10 milliseconds up to once per 27.77 hours (99999990 milliseconds) for the REST and MQTT Client
- Support for authentication and TLS / SSL encryption on all agents
- Support for user-level access based on the KEPServerEX User Manager and Security Policies Plug-In
- Configurable header and payload information for integration with different third party endpoints

Architectural Summary

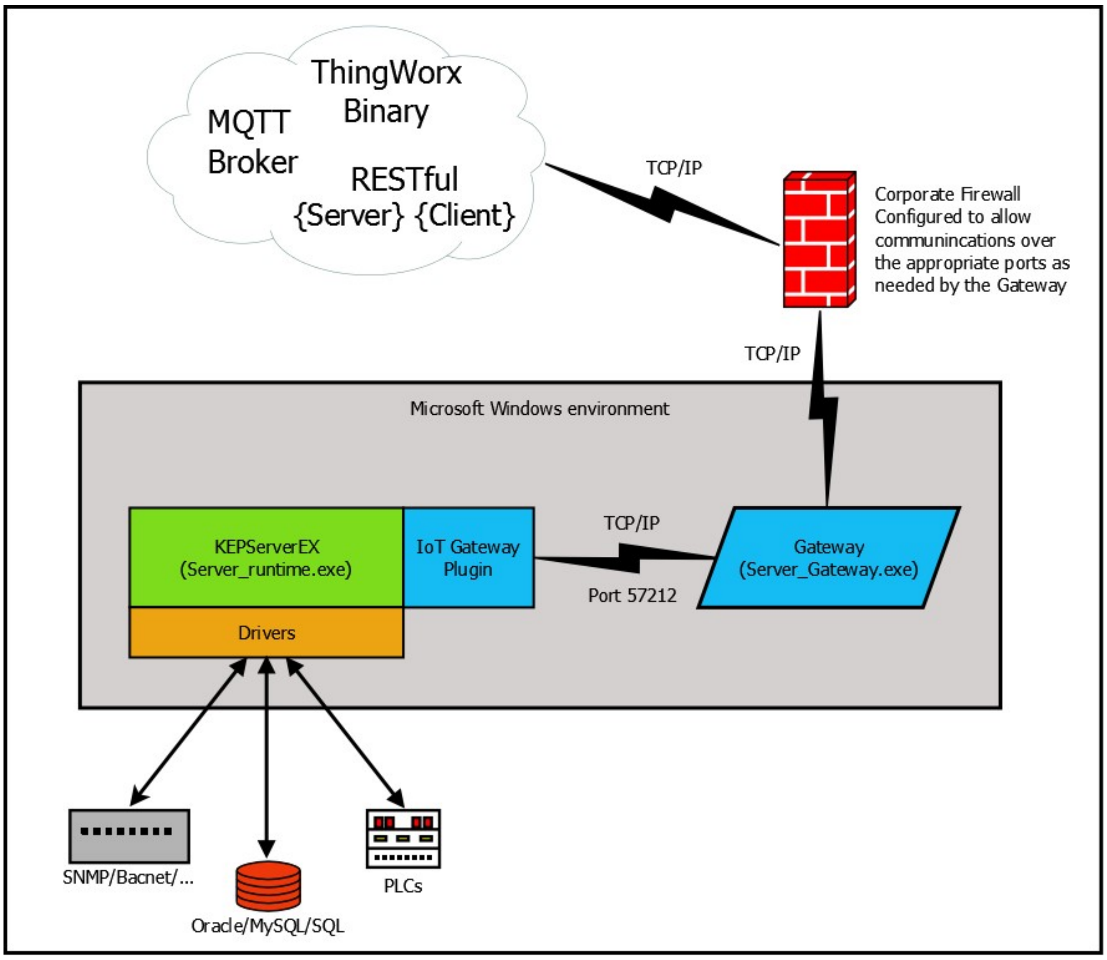
The IoT Gateway feature includes two main components:

- The server plug-in (IoT_Gateway.dll) is responsible for:
  - Configuration of the MQTT, ThingWorx, REST client and, REST server agents
  - Data collection from the server runtime

- Configuration of the Gateway settings
- License enforcement

- The IoT Gateway system service (server_iotgateway.exe), which:
  - Manages the connections to each third party endpoint
  - Buffers data collected from the plug-in
  - Provides the authentication and encryption layer to each agent

This diagram shows the layout of the IoT Gateway and components. The plug-in and gateway install on the same machine with KEPServerEX. KEPServerEX uses drivers to connect to data sources. That data is collected in the plug-in and sent to the gateway. The gateway publishes that data to the configured endpoint(s). In this diagram, data flows from the device/data sources at the bottom up to the endpoints at the top.



External Dependencies

For the IoT Gateway to run, KEPServerEX requires a working 32-bit Java JRE or full JDK installation version 7 or higher *(see note below)*. At this time, a 64-bit JRE or JDK is not supported. Kepware recommends the most current supported version of Java for use with the IoT Gateway. The current JRE may be downloaded and installed from Oracle at the following link:

https://java.com/en/download/

At the time of publication Java 8 with all updates has been tested and confirmed to be compatible.

Tip: Java does not need to be enabled in a browser for the gateway to run.

Notes:
Because the IoT Gateway is a product that has the potential to push data across the Internet to third-party endpoints, configuring the host computer or corporate firewall appropriately to allow just the specific ports that those endpoints are configured to use is recommended.

To prevent the loss of data and to keep KEPServerEX running properly, installing Java updates while in production is not recommended. The runtime service and gateway service should be taken offline before a Java update is run. Java 8 has changed the way that it updates, allowing multiple versions of Java to exist on the computer at the same time. With Java 8 a new version is placed side by side with the existing version. In this scenario, the default gateway behavior is to continue to use the old version of Java 8 until a reboot, an IoT Gateway restart or a change to the Java configuration in the Server Settings. If a JRE or JDK is specified to use for the IoT Gateway in the Server Settings, it will continue to use that version even after an update. The version of Java running is available through the Event log entries in KEPServerEX. Please see the online Knowledge Base for specific recommendations for updating Java.

General Operation

This section explains how the two components described above work together to form the basis of the IoT Gateway. This discussion also serves as an introduction to the terminology used in the remainder of this document.

Initialization

An agent configuration is created using the IoT Gateway from within the Server Configuration user interface. Details of this are covered later in the document. When an agent is configured and a runtime connection with the Server Configuration exists, the server_iotgateway service starts as directed by the Plug-in. At this time, the configuration is transferred from the Plug-in to the gateway where it is initialized. There may be multiple configurations for the same type of endpoint in the Plug-In. Each of these configurations creates its own instance on the gateway.

Startup

At system startup with a configured agent, the Server Runtime loads its project file (e.g. default.opf). Upon detecting that an agent is defined, the plug-in starts the server_iotgateway service. The plug-in establishes a connection to the gateway service and transmits the active agent configuration(s).

Data Updates

Data updates are managed by the plug-in for the REST, ThingWorx, and MQTT clients. The agent creates a server item reference from each configured tag and polls for data at the configured scan rate like any other client. Scan rates are configured on a per tag basis. The updates received are forwarded to the server gateway service, where they are buffered and eventually pushed to the third party endpoint at the configured publish rate.

Each data update persisted to the agent consists of four elements: ID, value, quality, and timestamp.

In the default “narrow format,” new data is pushed to the gateway when there is a change in value. More than one value change may be published per tag if more than one value update was received before the next publish time. All buffered data is sent with narrow format. The Every Scan feature sends an update to the gateway to be published to an endpoint for every good-quality scan of the tag whether or not there was a data change. A bad-quality scan will be sent only once. When the “every scan” radio button is selected on a tag, the deadband setting for that tag is ignored. The “Wide format” option sends only the last data update for every tag enabled on that agent on each publish whether there was a data update or not. No buffered data is sent with the “Wide format” option.

Data Buffer

Each individual agent has a buffer of 10,000 events in case the third-party endpoint is unreachable. An event is a single value change of a single tag that is configured on that agent. The buffer stores the oldest 10,000 events; when full, it drops new data coming in. This buffer is entirely in memory and is not written to disk. There is no buffer enabled when using the “wide format” publish on the MQTT and REST client agents.

Shutdown

When the Server Runtime receives a request to shutdown, the IoT Gateway is responsible for stopping data collection. After sending the final tag updates, the IoT Gateway uses the messaging interface to tell the server gateway service to close any active TCP/IP connections to third-party endpoints.

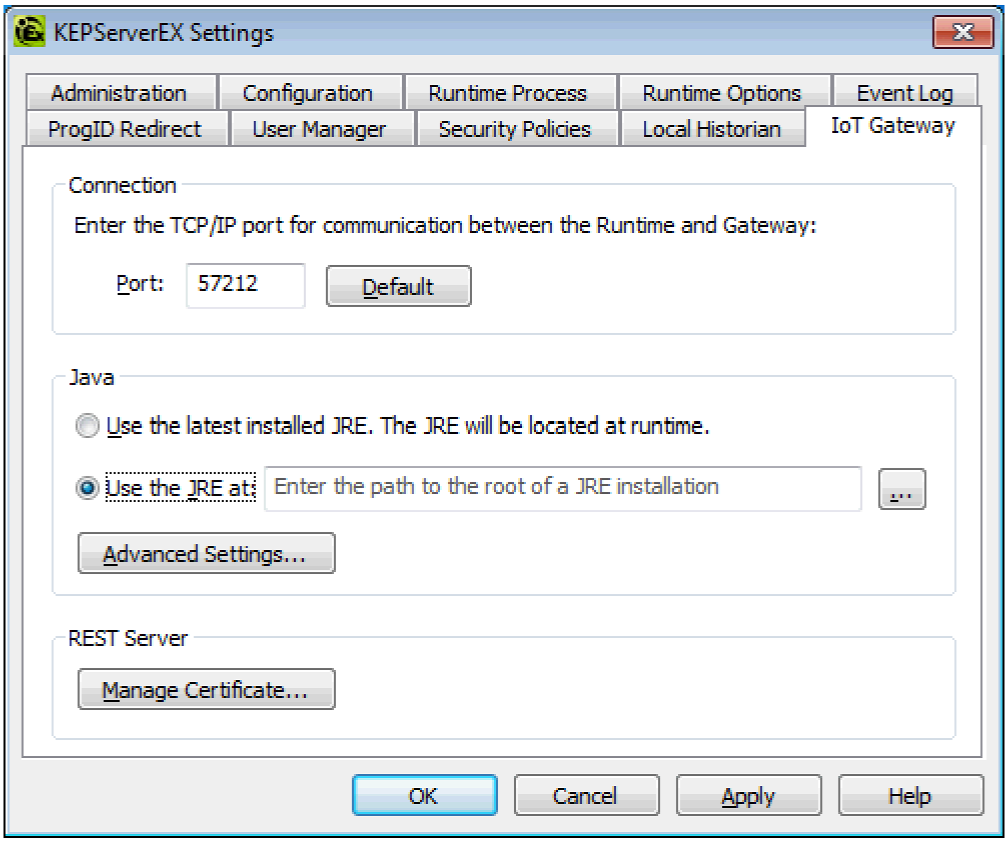
Configuration

Configuration of the IoT Gateway is accomplished in two places. The server_iotgateway service is configured from the Settings sections of the KEPServerEX Administration Icon in the system tray. This is where Java settings and gateway-level changes may be made. Generally these settings do not need to be adjusted for the IoT Gateway to function properly. The agents and tags themselves are configured from the IoT Gateway section of the KEPServerEX configuration window. Click any of the following for more information about configuration.

Configuring the Gateway

The IoT Gateway administrative settings are automatically configured on installation. If the settings need to be adjusted, access the IoT Gateway system settings by right-clicking on the Administration icon located in the system tray and selecting **Settings | IoT Gateway**.

Tip: If the Administrative icon is not in the system tray, re-launch it by selecting **Start | All Programs | Kepware | KEPServerEX 6 | KEPServerEX 6 Administration | Settings...**.


KEPServerEX Settings

Connection

Enter the TCP/IP port for communication between the Runtime and Gateway:

Port: 57212

Java

( ) Use the latest installed JRE. The JRE will be located at runtime.

[✓ Use the JRE at:] Enter the path to the root of a JRE installation

REST Server

In the Connection area:

Port: specifies the TCP/IP port that the server runtime and configuration use to communicate with the gateway service. The valid range is 1024 to 65535. The **Default** button populates the field with the default port number of 57212, configured by the server.

Tips:

1. The default port is recommended unless there is a conflict with another local application using that port.
2. Before changing the port setting, verify there is no conflict with the new port number.
3. The gateway service does not accept remote connections, so there should be no firewall implications associated with this port assignment.

In the Java area:

Use latest installed version of the JRE locates and utilizes the newest 32-bit JRE installed on the system when the IoT Gateway starts.

To specify a specific JRE, de-select this option and enter the path to the JRE or use the Browse **(...)** button to locate the JRE.

Tip: If **Use latest installed version of the JRE** is selected and the Java version is updated on the machine, the gateway service automatically starts using the updated version the next time the gateway is started. If this option is disabled, the gateway service continues to use the specified version.

Advanced Settings... allow Java-specific settings to be used. These settings should only be changed if instructed by Technical Support.

In the REST Server area:

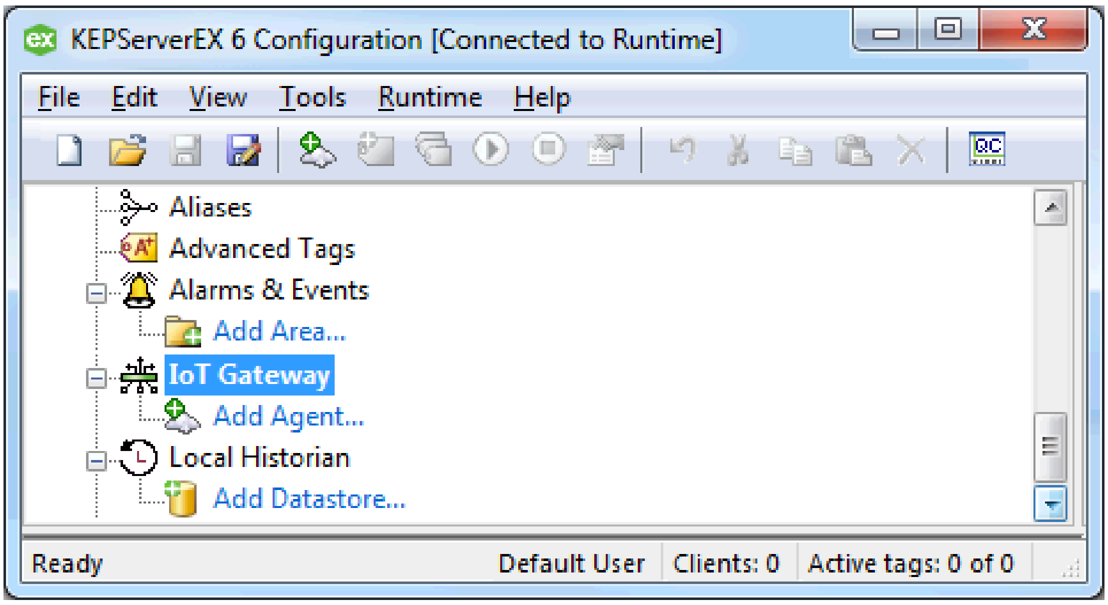
Click on the **Manage Certificate...** button to configure security certificate use for the REST server.

See Also:
Configure Gateway Certificate

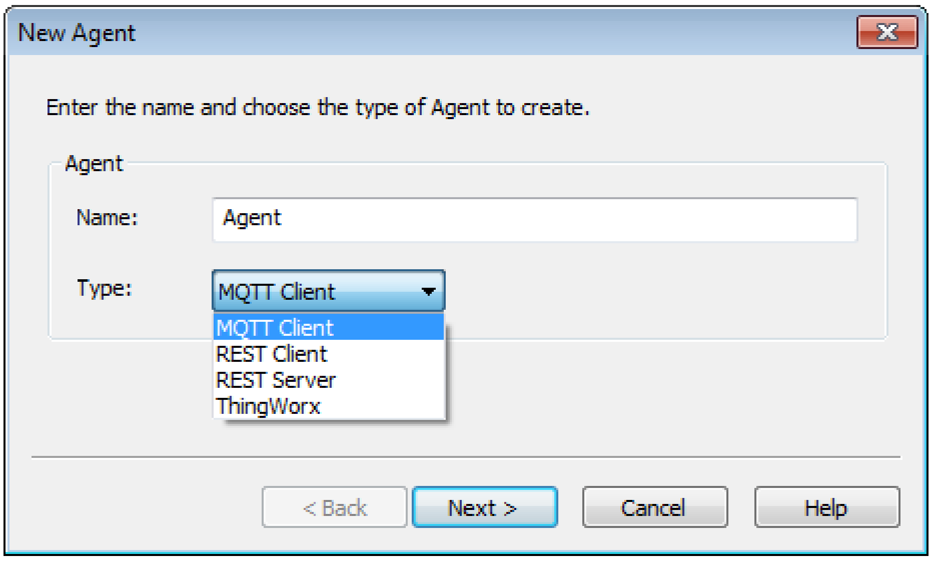
Configuring a New Agent

An agent configuration is required to begin publishing data to a third party endpoint. At least one agent needs to be configured with one active tag for the gateway service to start.

1. Click on the **Add Agent...** text or right-click any blank area in the agent pane and select **New Agent** from the pop up menu. Alternatively, click on the New Agent icon in the toolbar.



2. In the New Agent dialog, enter a name for the agent and select the type: MQTT Client, REST Client, REST Server, or ThingWorx agent.



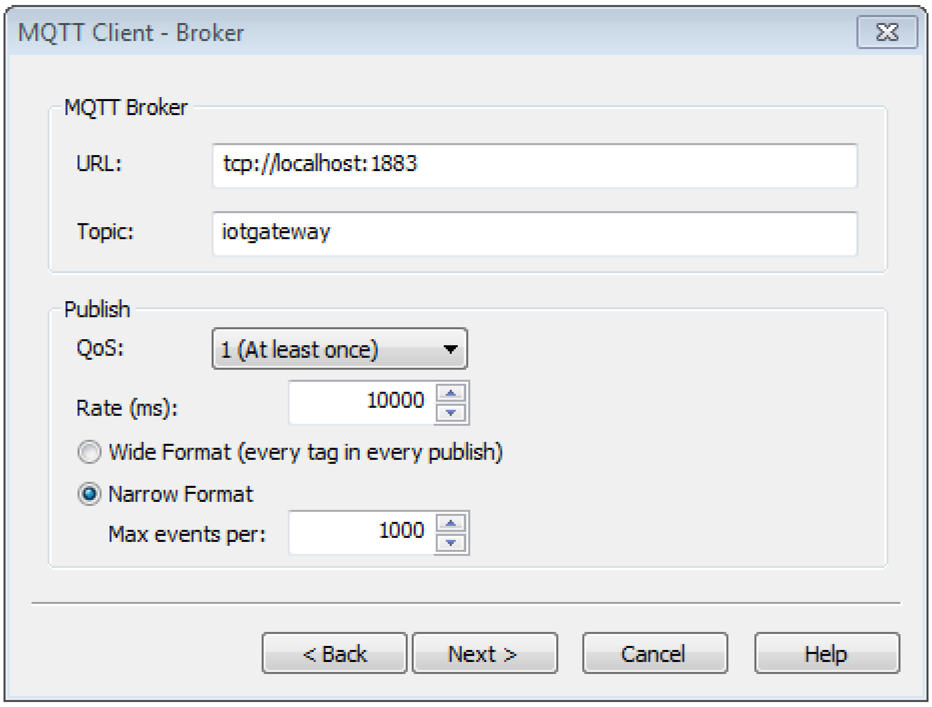
3. Click **Next >**.

4. Based on the type selected, proceed to:
   Configure an MQTT Client
   Configure a REST Client
   Configure a REST Server
   Configure a ThingWorx Connection

Configuring an MQTT Connection

Once the agent type is selected as an MQTT client, follow the steps below to create a new MQTT agent connection:

1. In the MQTT Client dialog, name the broker and configure its settings.


MQTT Client - Broker

MQTT Broker

URL: tcp://localhost:1883

Topic: iotgateway

Publish

QoS: 1 (At least once)

Rate (ms): 10000

Wide Format (every tag in every publish) [✓ Narrow Format]

Max events per: 1000

URL: the IP address or URL and port of the endpoint for the agent connection.

> **Tip**: If the endpoint uses an SSL connection, adjust the URL to use “ssl://” rather than “tcp://”.

Topic: the name used to filter or organize data published on the broker.

QoS: the MQTT setting for publishing data Quality of Service. Choices include: 0 (at most once), 1 (At least once), 2 (exactly once).

Wide Format (every tag in every publish): produces an output that includes all enabled tags in the agent with every scan regardless of value or quality changes. This format guarantees a consistent data format on every publish. Wide format sends only the latest value for each tag and has no buffer. If a publish fails while using wide format, the next publish is the latest scanned values for each tag.

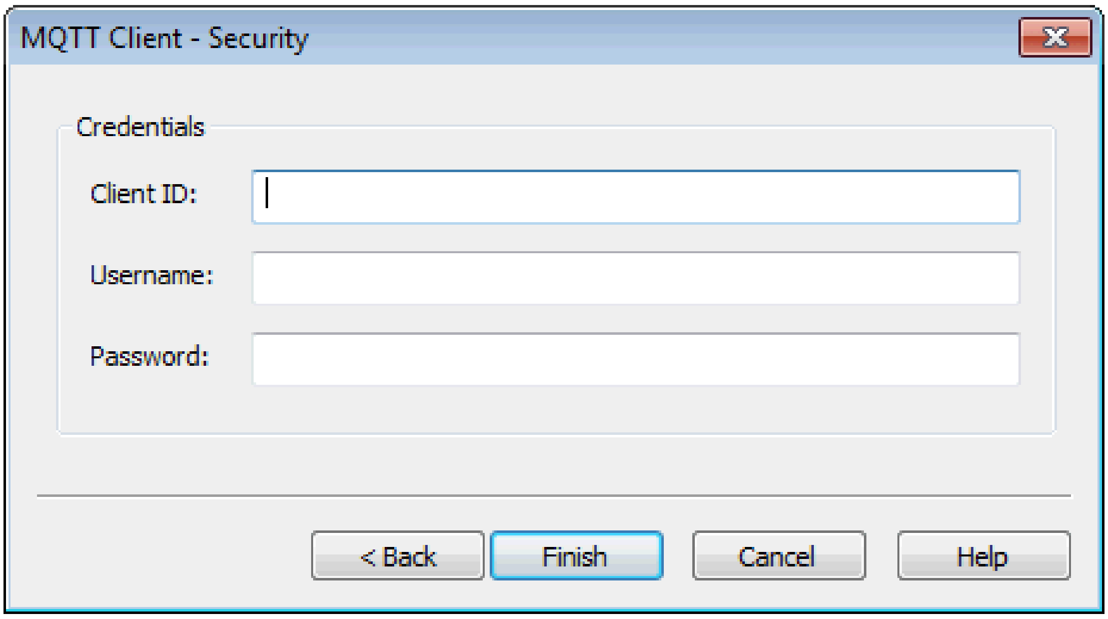
Narrow Format: produces output based on tags that have changed value or quality. This format buffers data and will publish all tag data changes to an endpoint.

Rate (ms): sets the frequency of agent data pushes to the endpoint. The range is from 10 to 99,999,990 milliseconds.

Max. events per: adjusts the number of tag events the gateway packages in a single trans-mission when using narrow format.

2. Click **Next >**.

3. In the MQTT Client - Security dialog, enter in the Client ID, Username, and Password for the broker being connected, if needed.



   Client ID: This is the unique identity for this agent's communication with the broker. Most brokers do not need a Client ID to connect.
   Username: This is the authorized user for authentication on the broker.
   Password: This is the password for basic authentication on the broker.
   Tip: This information is only needed if the broker requires it.
   Note: If not using an SSL encrypted connection, the username and password are sent as plain text to the broker. This is a limitation of the protocol.

4. Click **Finish**.

5. Add tags to the configuration.

6. The MQTT connection is now publishing to the broker. Verify the broker is receiving. If not, check the Event Log for errors.

See Also:
Adding Tags
Configuring Message
Security

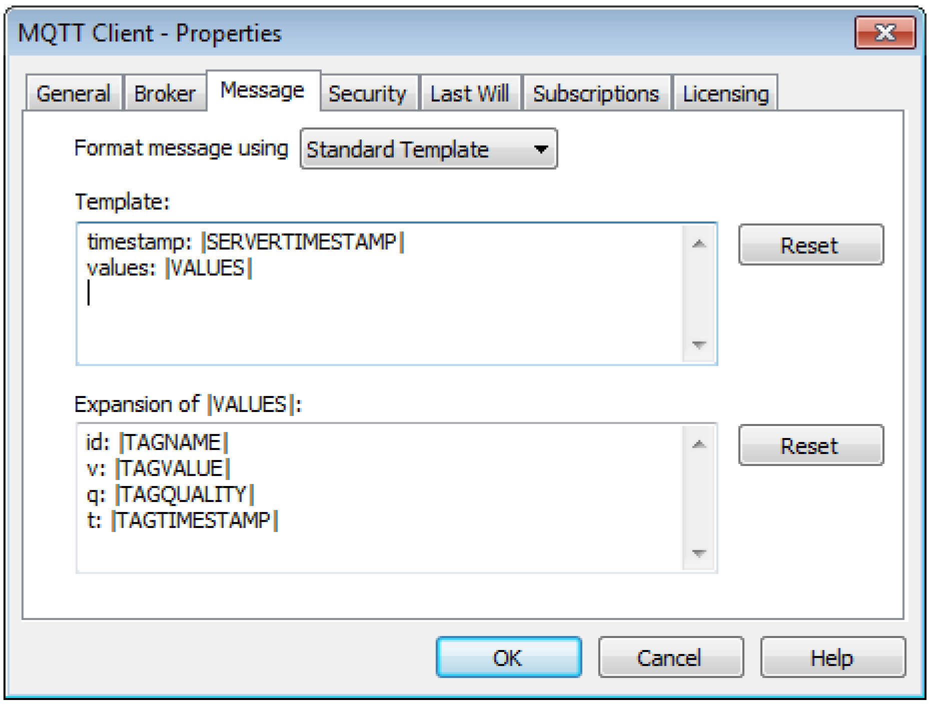
MQTT Client Message

To change the order of the default JSON data load or to remove data items, follow these steps:

1. Double-click on the agent name or right-click on the agent and select **Properties**.

2. From the Properties dialog, select the **Message** tab.

3. Make any desired changes (according the guidelines below) and click **OK**.



Format message using: Standard Template

Template:

```
timestamp: |SERVERTIMESTAMP|
values: |VALUES|
```

Expansion of |VALUES|:

```
id: |TAGNAME|
v: |TAGVALUE|
q: |TAGQUALITY|
t: |TAGTIMESTAMP|
```

Format message using: Select either **Standard Template** or **Advanced Template**.

Template: The template box contains any data to be sent before or after the JSON payload. This data is sent once per publish. Each variable may be preceded by any word or name, followed by a colon, then the variable. Key value pairs of static text may be included. Valid variables are:

> |SERVERTIMESTAMP| the time and date when the gateway published the data to the endpoint, in UNIX or POSIX time format
>
> |SERVERDATE| the same date and time as the timestamp, but in human-readable form
>
> |VALUES| the combined payload of the following box
>
> Click the **Reset** button to change the contents back to the defaults.

Expansion of |VALUES|: This box contains the format for the JSON payload delivered to the endpoint. These vales may be re-ordered or removed. A single publish event to the endpoint may include multiple instances of the data in this box. For example, if a tag had four data changes between publish events, there would be four complete JSON strings for that tag within this array. As with the Format box, each variable may be preceded by any desired word or descriptor. A colon is needed to separate the name from the variable. Valid variables are:

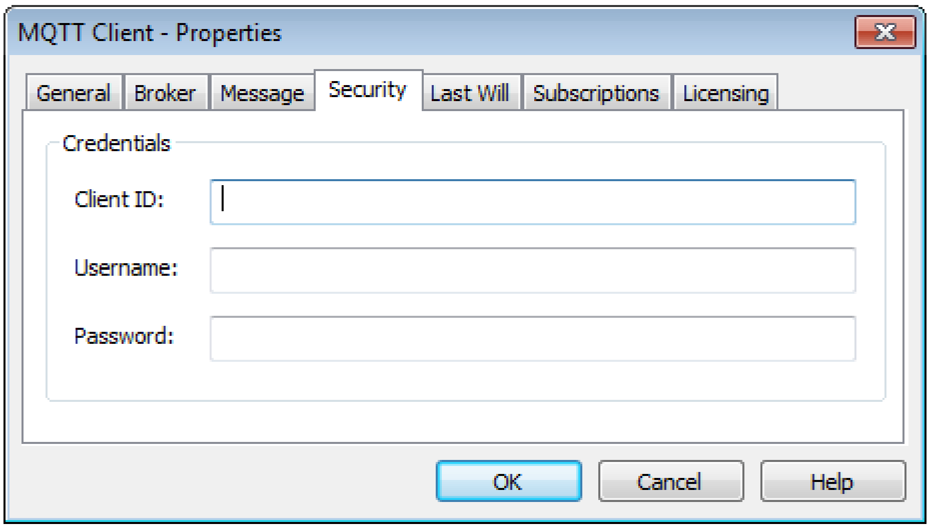
> |TAGNAME| The name of the selected tag
>
> |TAGVALUE| The value of the tag
>
> |TAGQUALITY| This denotes if the tag was read as good or bad
>
> |TAGTIMESTAMP| This is the time when the TAGVALUE was received
>
> Click the **Reset** button to change the contents back to the defaults.

Note: Please see the Advanced Template section for details on using that feature.

MQTT Client Security

Follow the steps below to configure or change MQTT agent authorization and access:

1. In the Security dialog, enter the credentials below



Client ID: the unique identity for this agent's communication with the broker. Most brokers do not need a Client ID to connect.

Username: the authorized user for authentication on the broker.

Password: the password for basic authentication on the broker.

Note: If not using an SSL encrypted connection, the username and password are sent as plain text to the broker. This is a limitation of the protocol.

2. Click **Finish**.
3. Proceed to **Add tags** to the configuration.
4. The MQTT connection is now publishing to the broker. Verify the broker is receiving. If not, check the Event Log for errors.

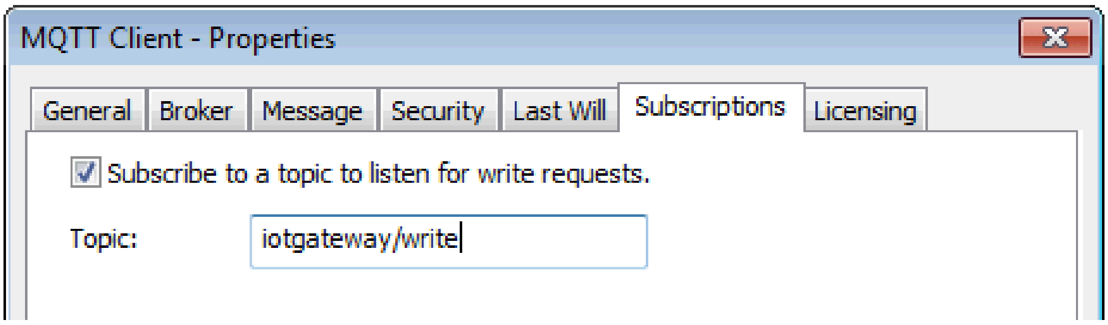
See Also:

Adding Tags

MQTT Subscriptions

The MQTT agent can be configured to subscribe to a topic on the broker, allowing other publishers to write to tags under that agent. When enabled, the agent listens to this topic with a QOS of 0 for properly formatted JSON to write to a tag.

1. Double-click on the agent name or right-click on the agent and select **Properties**.
2. From the Properties dialog, select the **Subscriptions** tab.



☑ Subscribe to a topic to listen for write requests.

Topic: iotgateway/write

3. Check **Subscribe to a topic to listen for write requests** to enable a subscription.
4. In the Topic field, enter the name of the topic to which the agent should subscribe.
5. Click **OK**.

The MQTT agent checks the topic specified for JSON data in the proper format. Once the agent parses the data, it attempts to write the value to the specified tag.
Only tags that have been added to the MQTT agent may be written to.

To perform a write, the data needs to be in the following format:

```
[{"id": "Channel1.Device1.Tag1","v": 42},{"id": "Channel1.Device1.Tag2","v": 523}]
```

The "Channel1.Device1.Tag1" should be replaced by the tag to be written and "42" by the value to be written. The example above shows a JSON array that should update both Tag1 and Tag2 when parsed by the MQTT agent.

Using Mosquitto_sub.exe to update Tag1 from a DOS command line would look like:

```
mosquitto_pub.exe -t iotgateway/write -m "[{\"id\": \"Channel1.Device1.Tag1\",\"v\":
42}]"
```

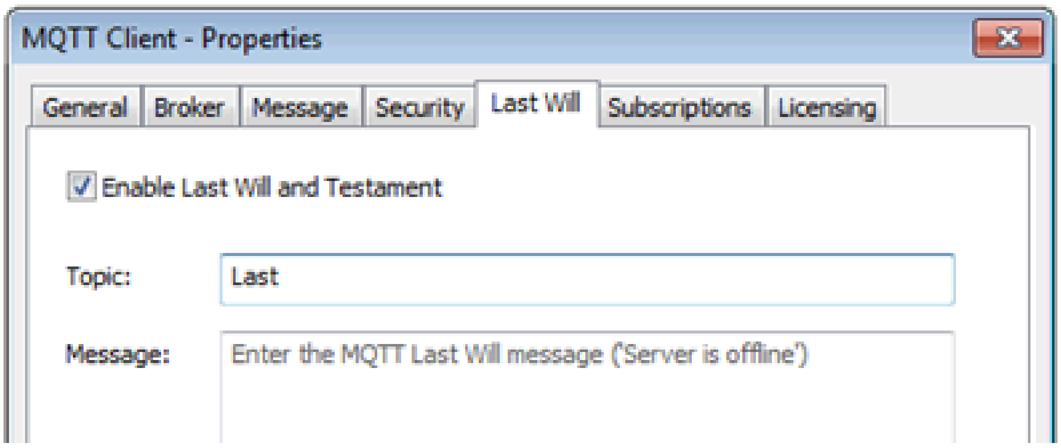
Tip: Any failures to update a tag using this method are posted to the KEPServerEX Event Log.

Note: The MQTT subscription option does not check for user authorization against the KEPServerEX User Manager or Security Policies Plug-In. Any valid JSON published to the configured topic will be written to the server. Configure the MQTT broker to verify that appropriate authentication is used at that level.

MQTT Last Will and Testament

The "Last Will and Testament" (LWT) is a convention for MQTT to notify subscribed clients when a client has disconnected unexpectedly without a "DISCONNECT" notice. To enable and configure an MQTT Last Will and Testament, follow these steps:

1. Double-click on the agent name or right-click on the agent and select **Properties**.
2. From the Properties dialog, select the **Last Will** tab.



☑ Enable Last Will and Testament

Topic: Last

Message: Enter the MQTT Last Will message ('Server is offline')

3. Check **Enable Last Will and Testament** to turn on a "last will" message.
4. Name the topic or path to serve as the last will and testament.
5. Enter the text that subscribed clients will receive in the last will and testament message. This is typically an explanation of the ungraceful disconnect, such as "offline" or "unexpected exit" to help the client.
6. Click **OK**.

In addition to the Last Will message, the MQTT agent publishes a message to the Last Will topic if the first data publish succeeds and when the MQTT agent is shut down due to a project edit, server reinitialize, or server shutdown.

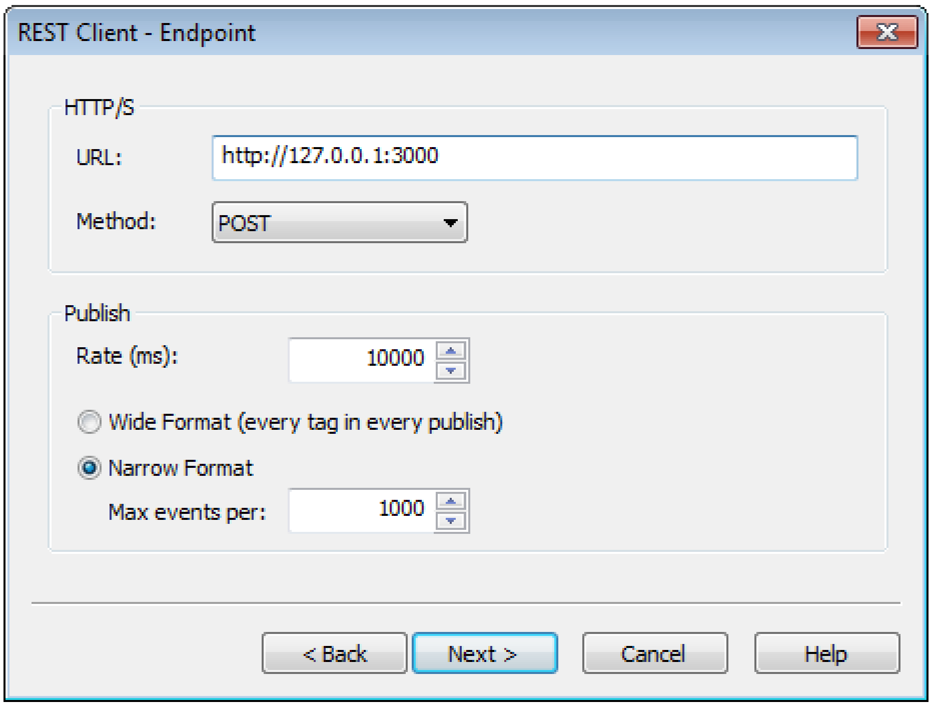
On first successful publish, the following text is published to the Last Will topic if the feature is enabled: "Server is online."

On graceful shutdown, the following text is published to the Last Will topic if the feature is enabled: "Server is shutting down."

Configuring a REST Client Connection

Once the agent type is selected a REST Client, follow the steps below to create a new REST Client agent connection

Define the REST Client Endpoint in the below fields, then click **Next >**.


REST Client - Endpoint

HTTP/S

URL: http://127.0.0.1:3000

Method: POST

Publish

Rate (ms): 10000

( ) Wide Format (every tag in every publish)

(●) Narrow Format

Max events per: 1000

URL: the IP address or URL and port of the endpoint for the agent connection. If the endpoint uses an SSL connection, adjust the URL in this box to use "https://"

Method: the way that the agent publishes data to the endpoint. It may be through a POST or PUT command.

Wide Format (every tag in every publish): produces an output that includes all enabled tags in the agent with every scan regardless of value or quality changes. This format guarantees a consistent data format on every publish. Wide format sends only the latest value for each tag and has no buffer. If a publish fails while using wide format, the next publish is the latest scanned values for each tag.

Narrow Format: produces output based on tags that have changed value or quality. This format buffers data and will publish all tag data changes to an endpoint.

Publish Rate: the frequency at which the agent pushes data to the endpoint.

Max. events per: adjusts the number of tag events the gateway packages in a single transmission when using narrow format.

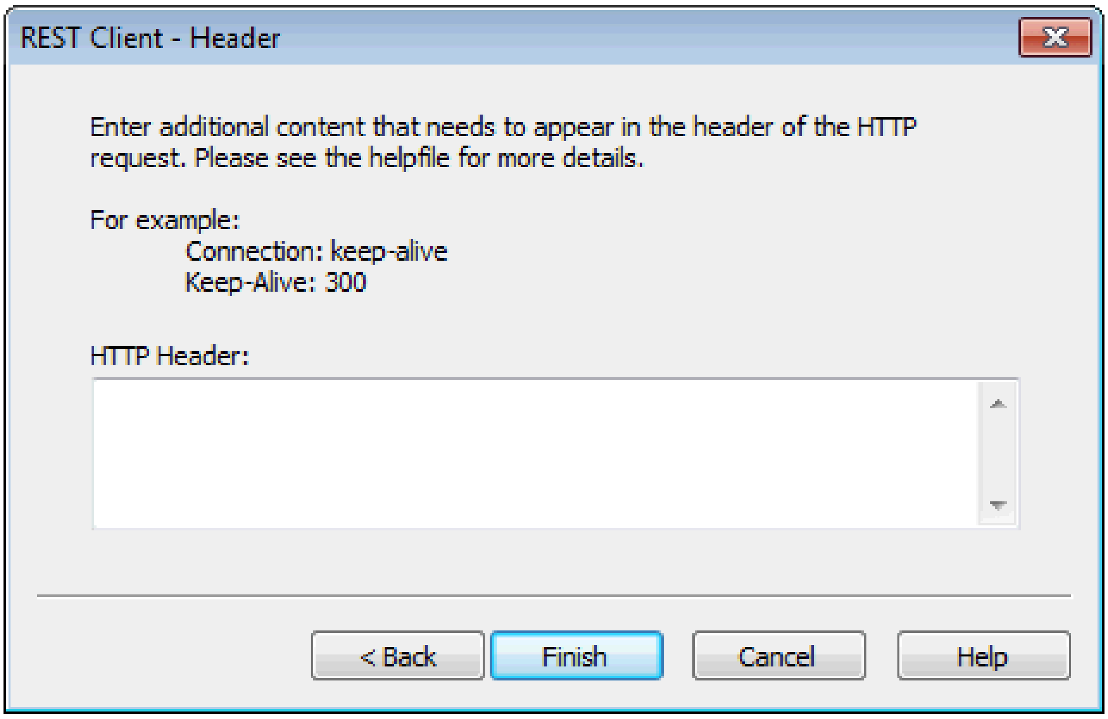
See Also:

Adding Tags
REST Data Header
REST Data Body
REST Security
Licensing

REST Client Header

Once the REST Client connection is created, the data header must be defined.

1. In the HTTP Header field, add name-value pairs to be sent to the REST server endpoint. This information is static and is sent with each connection to the endpoint.


REST Client - Header

Enter additional content that needs to appear in the header of the HTTP request. Please see the helpfile for more details.

For example:
Connection: keep-alive
Keep-Alive: 300

HTTP Header:

2. Click **Finish**.
3. A REST client agent has been added. Once tags are added to this client, it begins publishing to the endpoint as long as the agent is enabled.

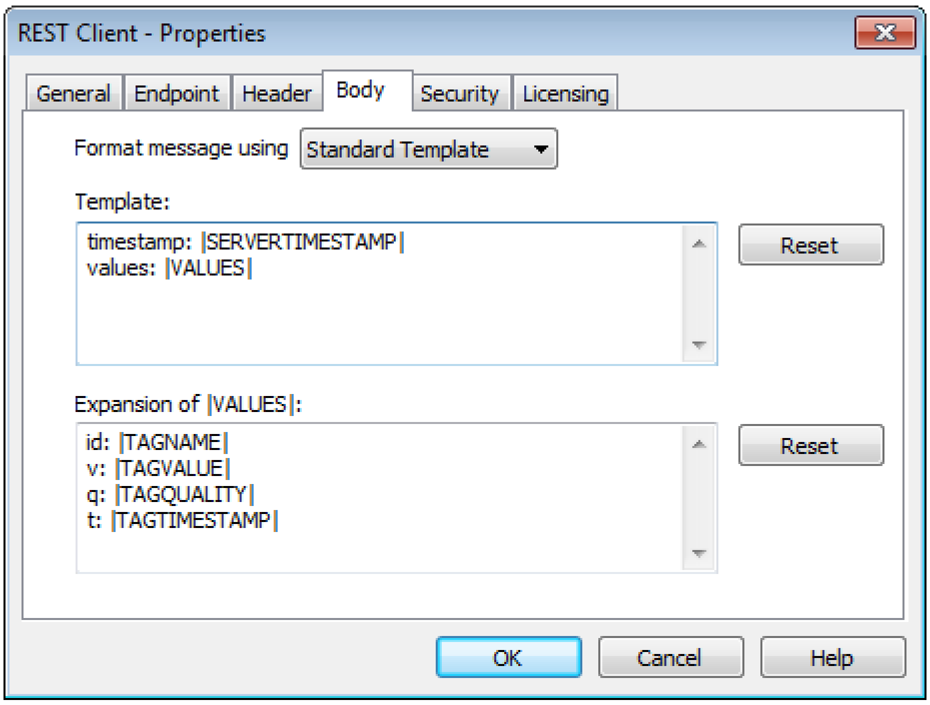
See Also:

Licensing

REST Client Body

To change the order of the JSON data load or to remove data items, follow these steps:

1. Double-click on the agent name or right-click on the agent and select **Properties**.
2. From the Properties dialog, select the **Body** tab.
3. Make any desired changes (according the guidelines below) and click **OK**.


REST Client - Properties

Format message using: Standard Template

Template:

```
timestamp: |SERVERTIMESTAMP|
values: |VALUES|
```

Expansion of |VALUES|:

```
id: |TAGNAME|
v: |TAGVALUE|
q: |TAGQUALITY|
t: |TAGTIMESTAMP|
```

Format message using: Select either **Standard Template** or **Advanced Template**.

Template: The template box contains any data to be sent before or after the JSON payload. This data is sent once per publish. Each variable may be preceded by any word or name, followed by a colon, then the variable. Key value pairs of static text may be included. Valid variables are:

> |SERVERTIMESTAMP| the time and date when the gateway published the data to the endpoint, in UNIX or POSIX time format
>
> |SERVERDATE| the same date and time as the timestamp, but in human-readable form
>
> |VALUES| the combined payload of the following box
>
> Click the **Reset** button to change the contents back to the defaults.

Expansion of |VALUES|: This box contains the format for the JSON payload delivered to the endpoint. These vales may be re-ordered or removed. A single publish event to the endpoint may include multiple instances of the data in this box. For example, if a tag had four data changes between publish events, there would be four complete JSON strings for that tag within this array. As with the Format box, each variable may be preceded by any desired word or descriptor. A colon is needed to separate the name from the variable. Valid variables are:

> |TAGNAME| The name of the selected tag
>
> |TAGVALUE| The value of the tag
>
> |TAGQUALITY| This denotes if the tag was read as good or bad
>
> |TAGTIMESTAMP| This is the time when the TAGVALUE was received
>
> Click the **Reset** button to change the contents back to the defaults.

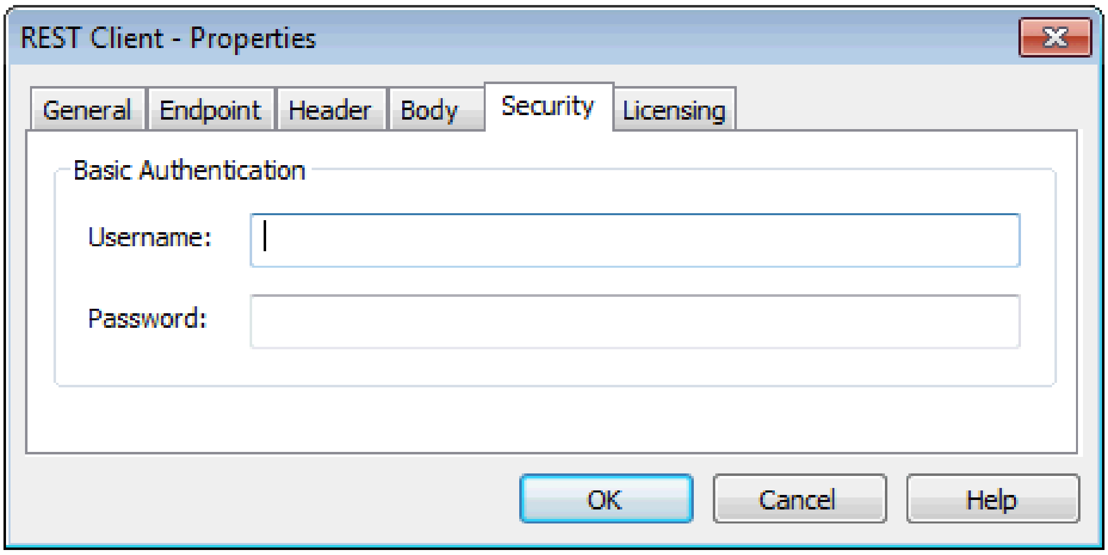
Note: Each expression in this box needs to include a name-value pair.

See Also: Advanced Template Data Format

REST Client Security

If the REST client endpoint needs basic authentication, follow these steps:

1. Double-click on the agent name or right-click on the agent and select **Properties**.
2. From the Properties dialog, select the **Security** tab.



Username: enter a valid user identity string for basic HTTP authentication.
Password: enter the password associated with the specified username.

3. Make the appropriate changes and click **OK**.

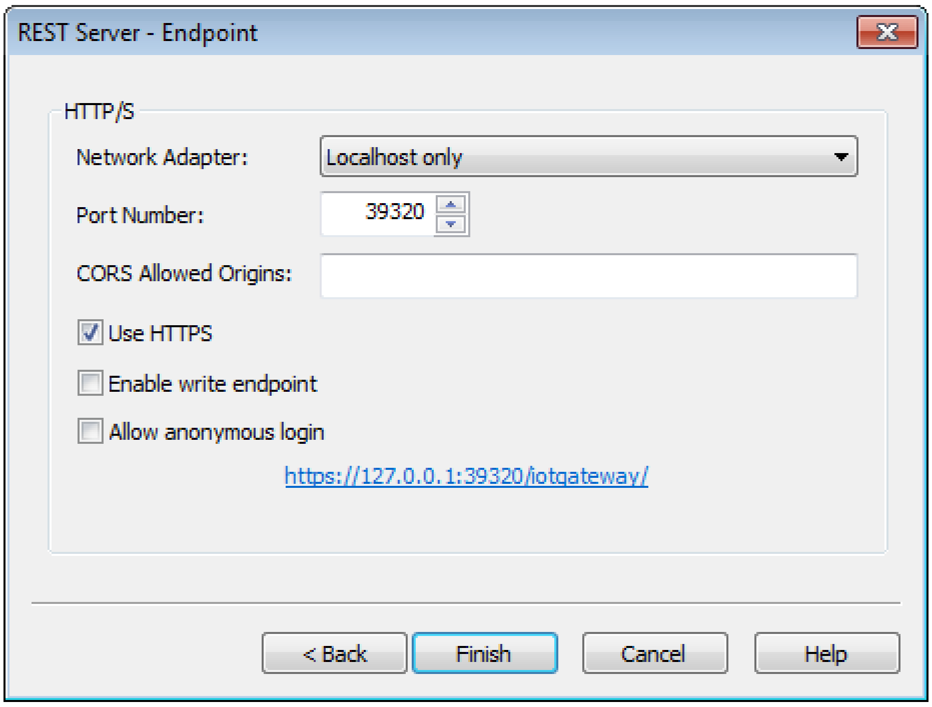
See Also:
Adding Tags
REST Data Body
Working with a REST Server
Licensing

Please also consult the documentation for KEPServerEX, User Manager, and Security Policies Plug-in.

Configuring a REST Server Connection

Once the agent type is selected as a REST Server, follow the steps below to create a new REST Server connection

1. In the Endpoint dialog, set the network adapter, the port, the endpoint, and security.



REST Server - Endpoint

HTTP/S

Network Adapter: Localhost only

Port Number: 39320

CORS Allowed Origins:

[✓] Use HTTPS

[ ] Enable write endpoint

[ ] Allow anonymous login

https://127.0.0.1:39320/iotgateway/

- **Network Adapter**: This sets the Ethernet connection where the REST server will respond. The default, **Localhost only**, has the server respond on either localhost or 127.0.0.1 and is only accessible from the computer where KEPServerEX is installed. The drop-down list includes the network cards configured for the local computer. Select a specific card for the REST server if there is more than one. For the REST server to respond on all network connections, select **Default**.
- **Port Number**: This is the port to which the REST server binds. If there are multiple REST server agents configured on the same network adapter, they each need a different port number.
- **CORS Allowed Origins**: This field allows entry of a comma-delimited list of allowed Cross-Origin Resource Sharing sources. A wildcard of * may also be used to accept any origin. The origins must be an exact, case-sensitive match of the origin sent by the HTTP client. When the field is empty, CORS is disabled, which means that pre-flight OPTIONS requests fail with HTTP error 403/FORBIDDEN and responses to non-preflighted requests will not have CORS headers appended to them.
  Note: This setting is only used when accessing the IoT Gateway REST Server from a custom web page.
- **Use HTTPS**: This function encrypts the data between the remote client and this REST server. Once installed, a self-signed certificate is created to allow this functionality. To use

a custom certificate, import a PFX file in the IoT Gateway section of the Server Administrator settings. This is enabled by default.

- **Enable write endpoint**: This allows or prevents the ability to write to any tags, regardless of the logged-in user's access level. When enabled, tags that are designated as read/write tags may be written based on user access level. If anonymous login is allowed, all accessing users may write to read/write tags. If anonymous login is not allowed, user credentials are respected regarding write permissions (based on the User Manager and or the Security Policy Plug-in). This options is disabled / unchecked by default.
- **Allow anonymous login**: By default, any client connection must have authentication credentials in the header that match a valid account in the User Manager or Security Policy Plug-in. If this option is enabled / checked, no look up for access is performed and connections are allow unauthenticated access. The User Manager and Security Policies Plug-in are both accessed from the server Administrator settings. This options is disabled / unchecked by default.
- The URL at the bottom of the dialog accesses the REST server. It is dynamic and changes as settings in this window change.

  Note: The live URL link shows the address once changes are applied. Clicking on the URL before changes are applied may result in a failure to load the page.

2. Once the setting are configured, click **Finish**.
3. Typically, **Adding Tags** is the next operation.
4. Begin **working with the REST server** by using standard HTTP browser requests, Browse and Read REST commands, formatted as:

   http://localhost:39320/iotgateway/browse

   http://localhost:39320/iotgateway/read?ids=<TagName>

See Also:
Adding Tags
Working with a REST Server
Licensing

Please also consult the documentation for KEPServerEX, User Manager, and Security Policies Plug-in.

Working with a REST Server

Once a REST server agent is created; a client may connect to the endpoint to browse, read, and write tags configured under that agent.

All REST server agent connections are checked against the User Manager to validate credentials unless the **Allow anonymous access** option is enabled on the agent.

For information about setting up the User Manager, refer to the KEPServerEX help guide. A quick review of the available commands may be found by using a web browser to navigate to the endpoint.

If the agent is configured with all the default selections, the link is https://localhost:39320/iotgateway/.

If the agent is not configured with the default selections, determine the specific link by opening the properties of the REST server agent and clicking on the **Endpoint** tab.

The IoT Gateway supports more than one REST server agent as long as they use different ports.

The REST server supports the following commands:

- Browse
- Read
- Write

The following GET commands may be tested in most web browsers.

Please note that current versions of Internet Explorer will no longer parse JSON in the browser and prompts to download it.

A Browse command will list all tags that are configured under this REST Server instance in a JSON array format. The format of the command is:
https://localhost:39320/iotgateway/browse

A Read command will return the tag or tags requested in a JSON array format. The format of the command is:
https://localhost:39320/iotgateway/read?ids=Channel1.Device1.Tag1

For multiple reads repeat the tags listing separated by &:
https://localhost:39320/iotgateway/read?ids=Channel1.Device1.Tag1&ids=Channel2.Device2.Tag2

The following are POST commands and will require the use of a more specialized client.

A Read post command will return the tag or tags requested in a JSON array format. The format of the command is:
https://localhost:39320/iotgateway/read

With the body of the POST containing the desired tags in the following format:

```
["Channel1.Device1.Tag1”]
```

Or for multiple tags:

```
["Channel1.Device1.Tag1","Channel2.Device2.Tag2","Channel3.Device3.Tag3"]
```

A write command allows a third-party client to write to one or more tags that are configured under the REST server agent. To be able to write to the tag, the agent must be configured to allow writes, and the tag must also be writable. The format of the command is:

https://localhost:39320/iotgateway/write

With the body of the POST containing the desired tag or tags and values in the following format:

```
[{"id": "Channel1.Device1.Tag1","v": "123"}]
```

For multiple tags:

```
[{"id": "Channel1.Device1.Tag1","v": "123"},{"id": "Channel2.Device2.Tag2","v": "456"},
{"id": "Channel3.Device3.Tag3","v": "789"}]
```

For the body, “id”: is the tag name and “v”: is the value write.

Tips:

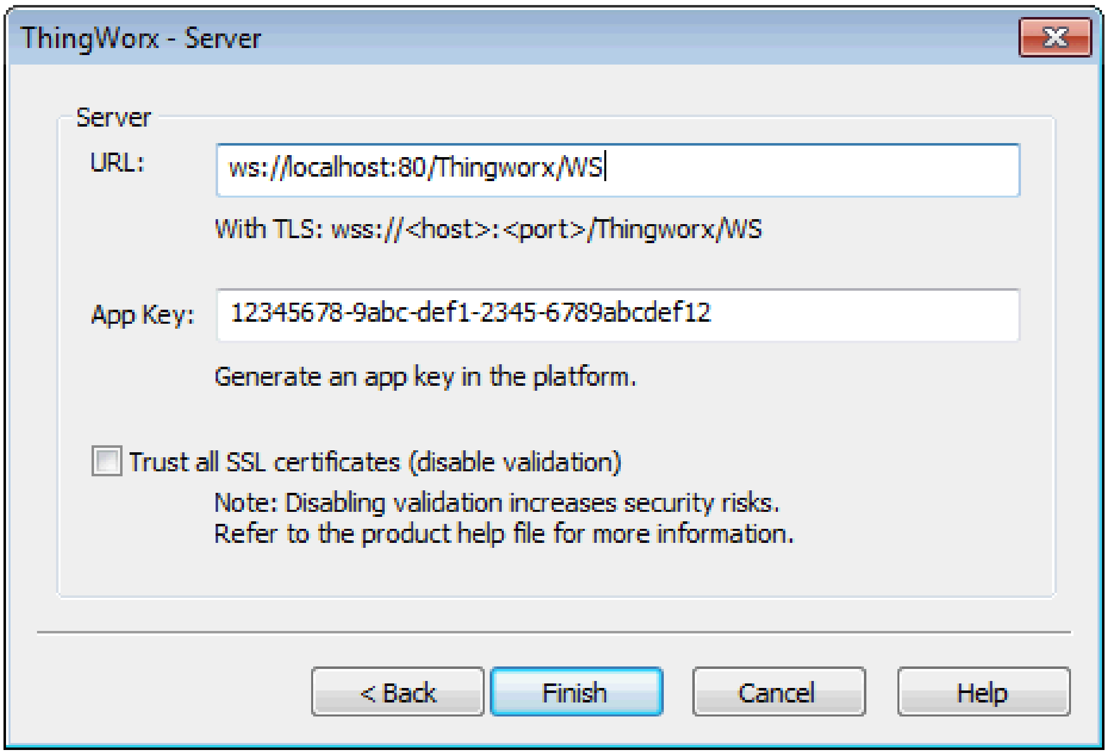
1. Directing a browser to https://localhost:39320/iotgateway/ or http://localhost:39320/iotgateway/; if HTTPS is disabled, provides a brief description of these commands.
2. When creating a custom web page to access data from the IoT Gateway REST server, it may be necessary to enable CORS or Cross-Origin HTTP requests. This is only needed for web pages; not custom REST clients. CORS is a security feature of modern browsers that verifies that data displayed on a page originates from a valid source. Using the * character in the CORS Allowed Origin field ensures that requests from any origin succeed. Leaving the field blank results in an HTTP 403 FORBIDDEN response. To restrict access, determine the origin header from the location running the web page (information that can be found when using the developer web console of the browser) and enter that into the CORS Allowed Origins field.

Notes:

1. The port and tag names in these examples must match those configured in the REST server settings.
2. Computer names with underscores are not seen as valid endpoints and result in HTTP 500 errors.

Configuring a ThingWorx Connection

Once the agent type is selected as ThingWorx, follow the steps below to create a new connection. The agent name entered is the “Thing” name that must be created in the ThingWorx Composer.


ThingWorx - Server

Server

URL: ws://localhost:80/Thingworx/WS

With TLS: wss://<host>:<port>/Thingworx/WS

App Key: 12345678-9abc-def1-2345-6789abcdef12

Generate an app key in the platform.

☐ Trust all SSL certificates (disable validation)

Note: Disabling validation increases security risks.
Refer to the product help file for more information.

1. In the Server dialog, configure the ThingWorx endpoint URL, App Key, and certificate security.
   - **URL**: This is the URL where the ThingWorx endpoint is hosted. Starting the URL with ws creates a standard web socket connection to that endpoint. Starting the URL with wss uses TLS encryption to create a secure web socket connection with the ThingWorx endpoint. The ThingWorx endpoint must be configured to support a secure socket connection for this feature.
   - **App Key**: This is the authentication method used by ThingWorx. The App Key is generated in the ThingWorx Composer and must be added here to allow the ThingWorx agent to publish data.
   - **Trust all SSL Certificates** (disable validation): Enabling this option skips any SSL certificate check and assumes the certificate is valid. This option allows the use of self-signed certificates, such as when an SSL certificate is received from the ThingWorx server; it is not validated against a certificate authority. Disabling validation can lead to an insecure connection and should be done with caution.
2. Once the settings are configured, click **Finish**.
3. Typically, Adding Tags is the next operation.

   Note: When creating a new ThingWorx agent, an initial update of all tags is sent to the endpoint. This allows binding tags in the Composer. If the agent name is changed and or the endpoint URL is changed, only buffered and new data is sent to the endpoint. This may lead to tags not appearing under the Manage Binding section of the ThingWorx Composer. Setting the tag option “Send every scan” forces data to be pushed to the ThingWorx platform, allowing that tag to be added under the

Manage Binding section. Re-initializing the server causes an initial update of values to be sent to the ThingWorx endpoint.

4. Once complete, unbound the Thing within ThingWorx with the name of the new agent appears. Creating a Thing in the Composer with the Agent name allows direct management of tags as properties.

Note: Under the "Source/Remote Name" listing in Manage Bindings in the ThingWorx Composer, tag addresses have periods replaced with underscores. The "Local Name" in that section may be changed to anything compliant with ThingWorx Composer character rules and limits.

Changing an Agent Configuration

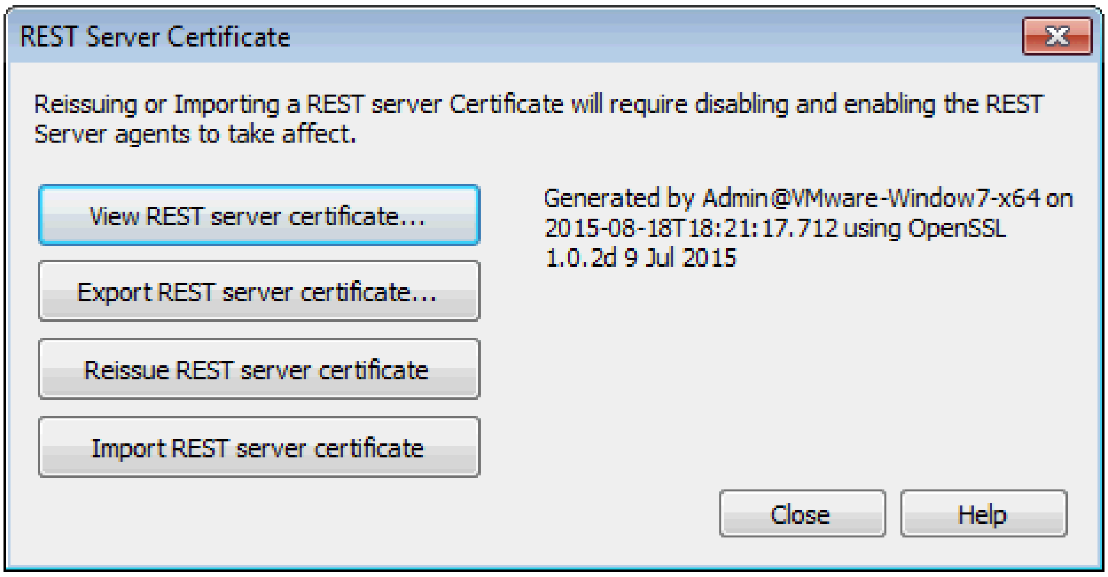
Agent settings can be updated after configuration. To access the settings, double-click the agent name or right-click on the agent and select Properties. Changes take effect immediately once submitted. This causes the gateway to reload the agent configuration.

Notes:

1. If there are any events in the agent's buffer when a property change is made, those events are not lost; they are pushed to the updated configuration.
2. Disabling an agent causes its buffer to be dropped.
3. If the endpoint URL is changed after the connection is created, no new initial update is sent. Only buffered values or new values are delivered to the new endpoint.

Configuring a Gateway Certificate

Through the **Administration | Settings...** menu; a certificate for the gateway can be viewed, exported, imported, or reissued.


REST Server Certificate
View REST server certificate...
Export REST server certificate...
Reissue REST server certificate
Import REST server certificate
Close
Help

Reissuing or Importing a REST server Certificate will require disabling and enabling the REST Server agents to take affect.

Generated by Admin@VMware-Window7-x64 on 2015-08-18T18:21:17.712 using OpenSSL 1.0.2d 9 Jul 2015

View REST server certificate... This displays the details of the current certificate.

Export REST server certificate... Use this button to save the current certificate in a .DER format for importing into third-party REST clients.

Reissue REST server certificate This creates a new certificate, replacing the current certificate.

Import REST server certificate... Use this button to import a certificate in .PFX format, only necessary using a custom-created certificate.

Note: When reissuing or importing a certificate, the new certificate does not take effect until the REST server endpoint(s) are stopped and restarted by disabling and re-enabling them or by reinitializing the server runtime.

See Also:
Configuring the Gateway

Configuring a Self-Signed Certificate

The IoT Gateway supports the use of self-signed certificates with the MQTT and REST clients. These agents use the Microsoft Windows, computer-level, trusted certificate store to keep track of these certificates. By using this store, most recognized certificate authorities are already approved. To import a certificate, use the below instructions.

Note: It is necessary to log in to the computer with an account that is part of the Local Administrators user group to add certificates to the appropriate Windows certificate store.

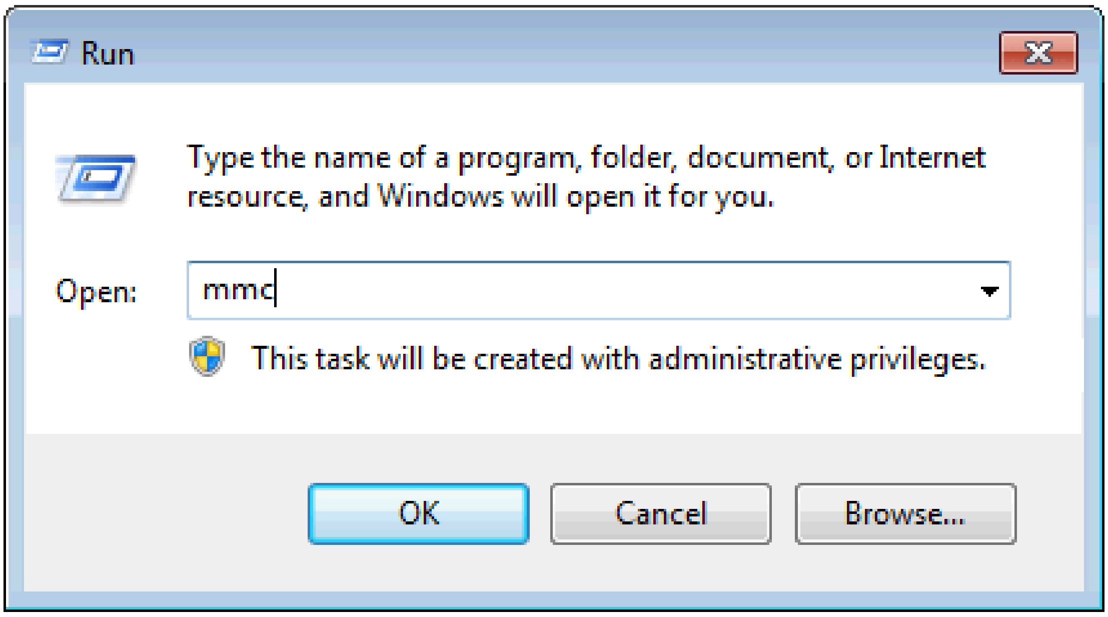
Command Line Steps

1. From the **Start** menu, select **All Programs**.
2. Choose **Accessories** then right-click on **Command Prompt** and select **Run as Administrator** from the menu.
3. In the command prompt window, navigate to the location of the certificate.
4. Enter the command:
   ```
   certutil -addstore "Root" <CertificateName>
   ```
   where the <CertificateName> is the name of the .cer or .crt file.
5. Press Enter to execute the command.
6. Verify the import is complete when several lines of output appear ending with:
   ```
   CertUtil: -addstore command completed successfully.
   ```

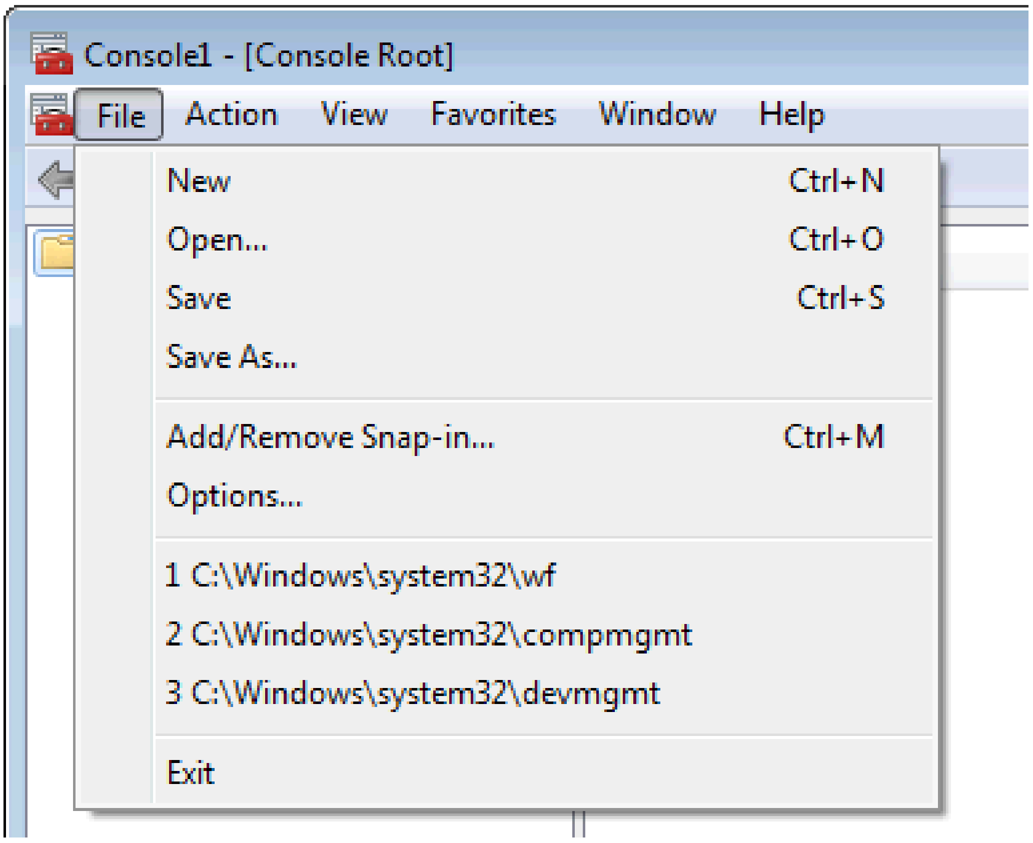
Windows Console Steps

1. From the **Start** menu, select **All Programs**.
2. Choose **Accessories**| **Run**.

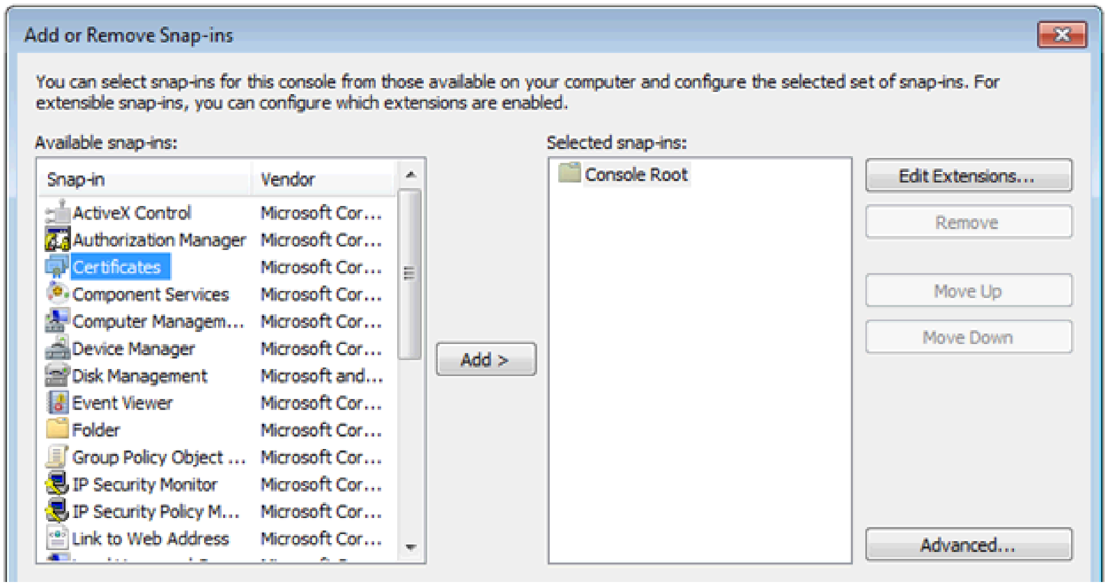
3. In the Run box, type “mmc” and click **OK**.


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
mmc
This task will be created with administrative privileges.
OK
Cancel
Browse...

4. In the console window, choose **File | Add/Remove Snap-in...**



5. Select **Certificates** on the left.


Add or Remove Snap-ins

You can select snap-ins for this console from those available on your computer and configure the selected set of snap-ins. For extensible snap-ins, you can configure which extensions are enabled.

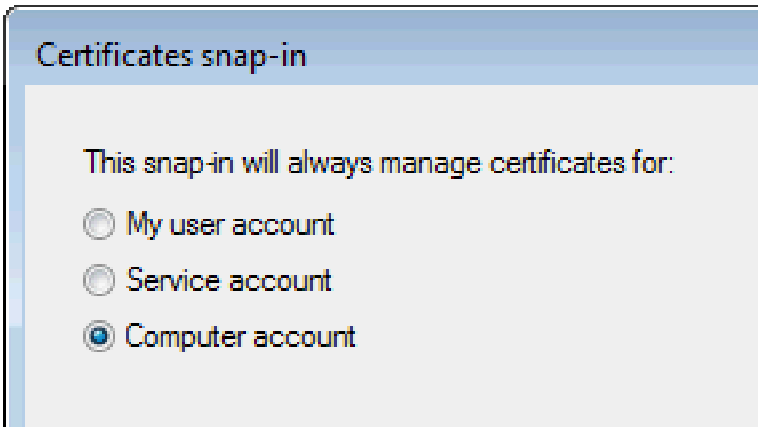
Available snap-ins:

Snap-in	Vendor
ActiveX Control	Microsoft Cor...
Authorization Manager	Microsoft Cor...
Certificates	Microsoft Cor...
Component Services	Microsoft Cor...
Computer Managem...	Microsoft Cor...
Device Manager	Microsoft Cor...
Disk Management	Microsoft and...
Event Viewer	Microsoft Cor...
Folder	Microsoft Cor...
Group Policy Object ...	Microsoft Cor...
IP Security Monitor	Microsoft Cor...
IP Security Policy M...	Microsoft Cor...
Link to Web Address	Microsoft Cor...

Selected snap-ins:

Console Root

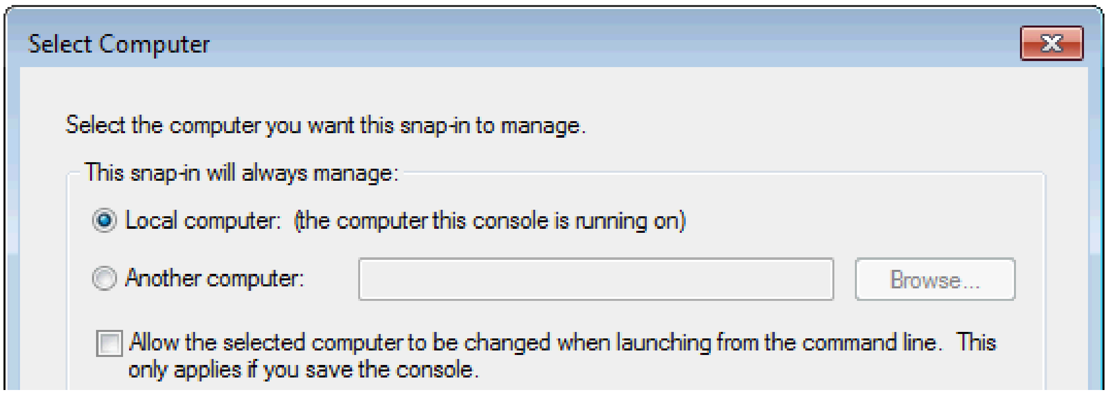
6. Click the **Add** button.


Certificates snap-in

This snap-in will always manage certificates for:

- My user account
- Service account
- [✓ Computer account]

7. Select **Computer account** and then click **Next >**.
8. Select **Local Computer** and click **Finish**.


Select Computer

Select the computer you want this snap-in to manage.

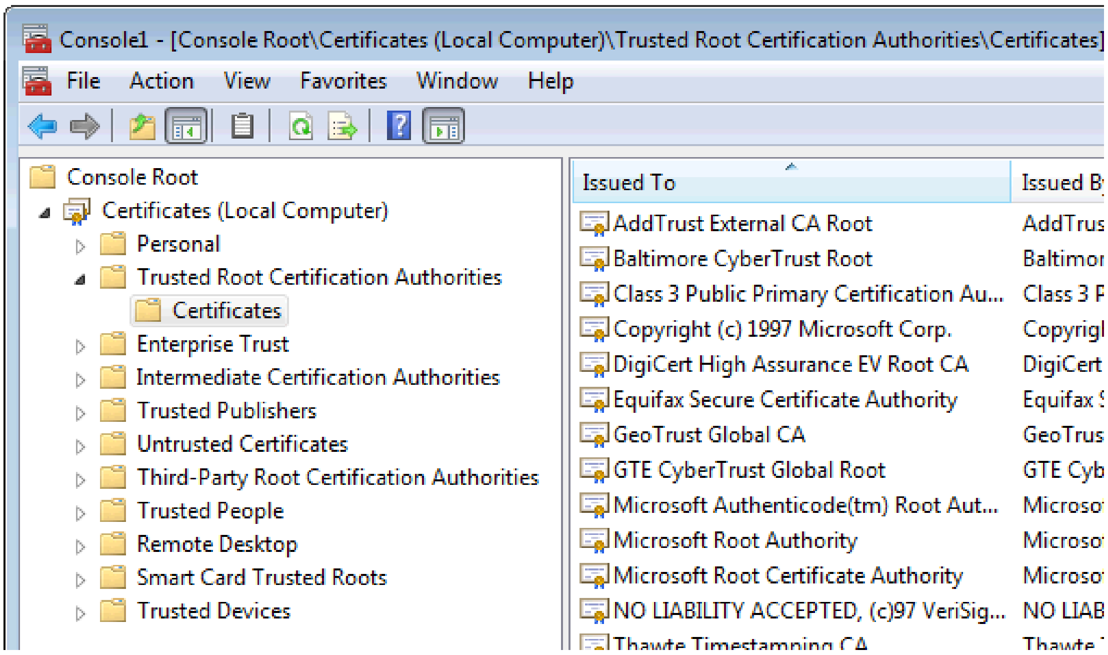
This snap-in will always manage:

- [✓ Local computer: (the computer this console is running on)]
- Another computer:

☐ Allow the selected computer to be changed when launching from the command line. This only applies if you save the console.

9. Back in the Add or Remove Snap-ins window, click **OK**.

10. Verify there is a Certificates (Local Computer) listing in the Console window.


Console1 - [Console Root\Certificates (Local Computer)\Trusted Root Certification Authorities\Certificates]
File
Action
View
Favorites
Window
Help
Console Root
Certificates (Local Computer)
Personal
Trusted Root Certification Authorities
Certificates
Enterprise Trust
Intermediate Certification Authorities
Trusted Publishers
Untrusted Certificates
Third-Party Root Certification Authorities
Trusted People
Remote Desktop
Smart Card Trusted Roots
Trusted Devices

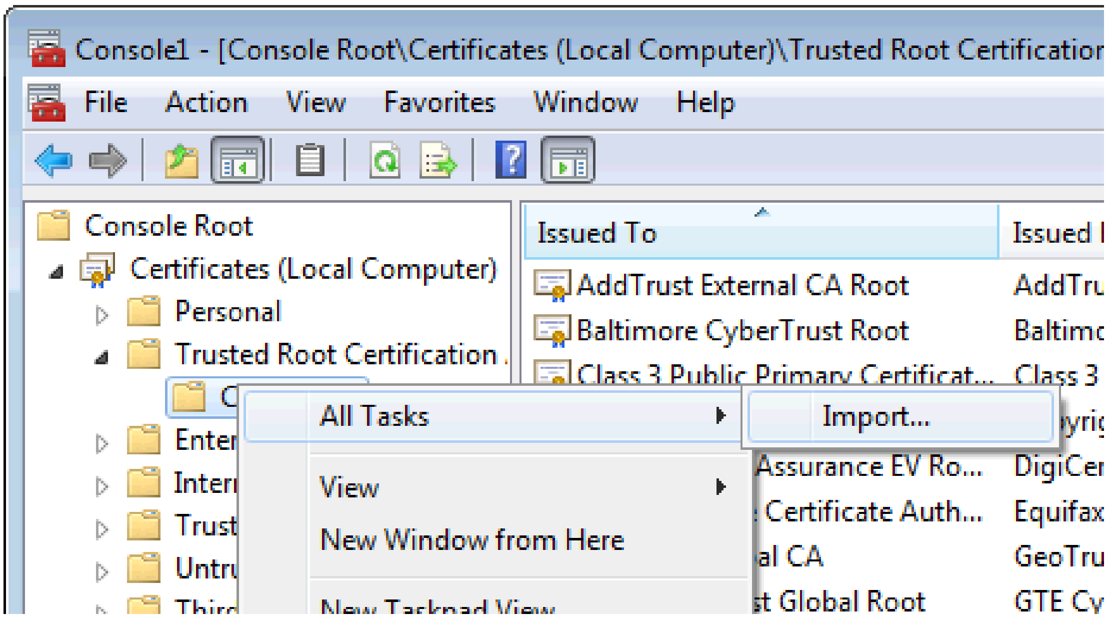
Issued To	Issued B
AddTrust External CA Root	AddTrus
Baltimore CyberTrust Root	Baltimor
Class 3 Public Primary Certification Au...	Class 3 P
Copyright (c) 1997 Microsoft Corp.	Copyrigl
DigiCert High Assurance EV Root CA	DigiCert
Equifax Secure Certificate Authority	Equifax S
GeoTrust Global CA	GeoTrus
GTE CyberTrust Global Root	GTE Cyb
Microsoft Authenticode(tm) Root Aut...	Microso
Microsoft Root Authority	Microso
Microsoft Root Certificate Authority	Microso
NO LIABILITY ACCEPTED, (c)97 VeriSig...	NO LIAB
Thawte Timestamping CA	Thawte

11. Expand the Certificates listing, then expand Trusted Root Certification Authorities.

12. Click on Certificates and verify a listing of all the Root certificates appears.

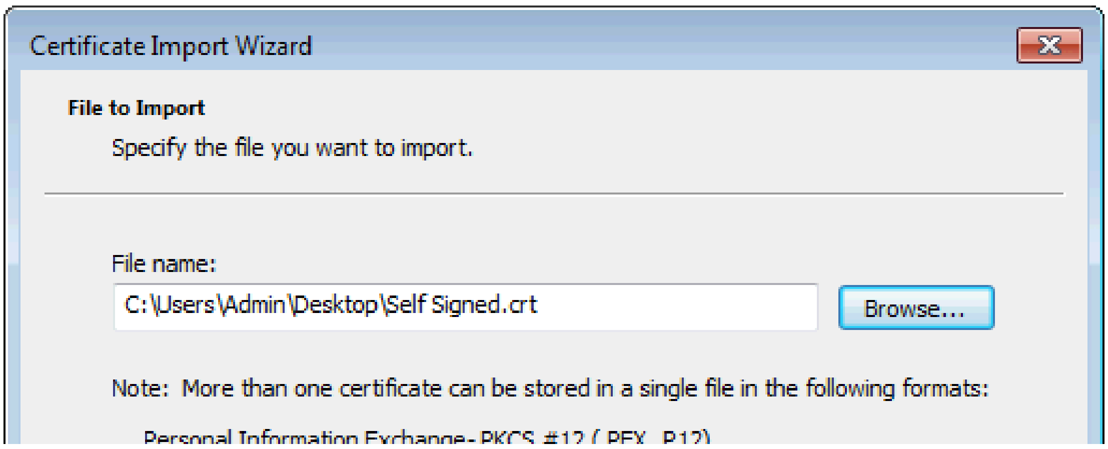
13. Locate the self-signed certificate to import for the MQTT and REST client.

14. Right-click on Certificates and select **All Tasks | Import...**


Console1 - [Console Root\Certificates (Local Computer)\Trusted Root Certification
File
Action
View
Favorites
Window
Help
Console Root
Certificates (Local Computer)
Personal
Trusted Root Certification
All Tasks
Import...
View
New Window from Here
New Taskpad View
Issued To
AddTrust External CA Root
Baltimore CyberTrust Root
Class 3 Public Primary Certificat...

15. In the Certificate Import wizard, click **Next** on the initial page.

16. Click **Browse...** to locate and select the certificate to import (most client certificates are in a .cer or .crt format).


Certificate Import Wizard

File to Import

Specify the file you want to import.

File name:

C:\Users\Admin\Desktop\Self Signed.crt

Note: More than one certificate can be stored in a single file in the following formats:

Personal Information Exchange- PKCS #12 (.PFX, .P12)

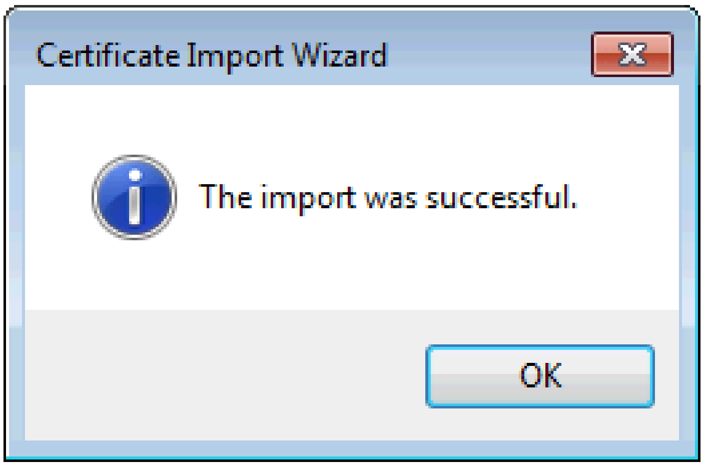
17. Click **Next >**.

18. Verify that **Place all Certificates in the following store** is selected and that store is the Trusted Root Certification Authorities.

19. Click **Next >**.

20. On the final page of the wizard, click **Finish**.

21. A pop-up message will confirm if the import was successful. Click **OK**, and close the Console window.


Certificate Import Wizard

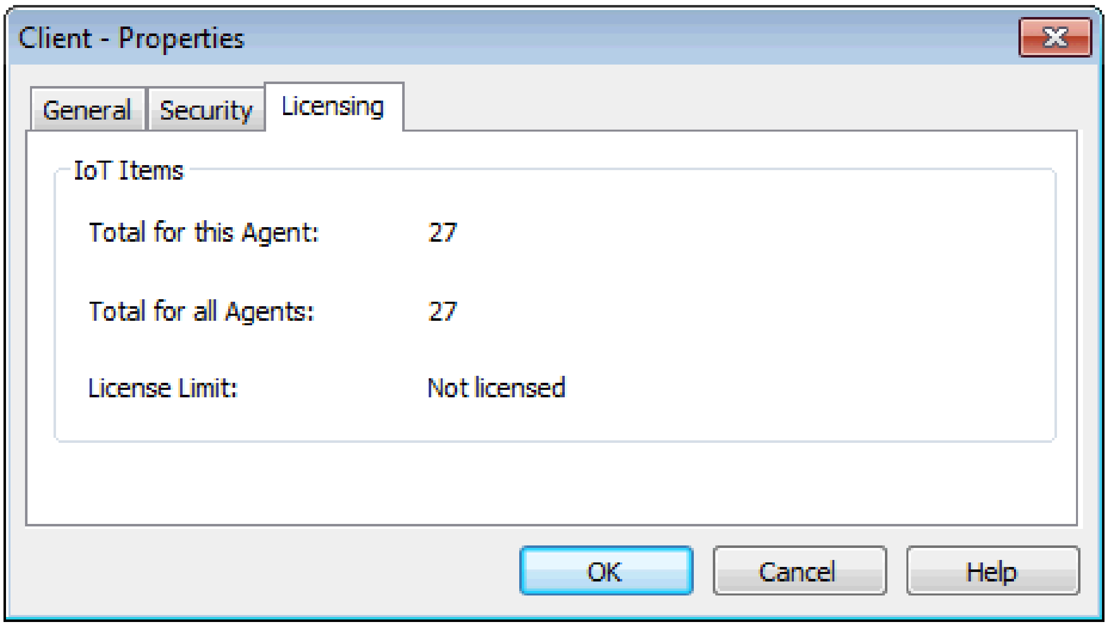
The import was successful.

Licensing

The IoT Gateway uses a tiered, count-based licensing model. A license may be purchased that enables the product to run for an unlimited amount time for a fixed maximum number of tags. The license limit does not prevent the addition of new tags beyond the tag count, nor does it signal the product to enter Demo Mode, but it does prevent updates from any tags added beyond that count.

The number of tags licensed is shown on the Licensing tab under the properties of each agent. There is a listing for the number of tags for that agent, as well as the total number of tags licensed in the IoT Gateway and the license limit.

The licensing tab lists the total number of tags configured under this agent as well as the total number configured under all the agents and the license level. Each agent has a licensing tab under properties. In this example, the client is the only agent configured and it has 24 tags configured in the tag window.



Total for this Agent: 27

Total for all Agents: 27

License Limit: Not licensed

Exceeding the Limit

An event log message is posted each time a new tag is created in excess of the license limit. If the license limit is exceeded, existing tags can be deleted to make license counts available for the new ones.

Notes:

1. When the license limit is exceeded, IoT Gateway processes the licensed number of tags only; the exact tags can vary based on the project loading order. The Event Log messages indicate the exact tags that did not load.
2. For licensing, tags are counted on a per-agent, not per-tag basis. For example; if the same tag is added to both a REST client and an MQTT client, it counts as two tags against the license.

Unlicensed Operation

When no license is installed, the entire product enters Demo Mode and runs for two hours before the runtime service must be stopped and restarted, which is accomplished through the administration icon found in the system tray.

Data

The IoT Gateway pushes data in a standard JSON format via the REST and MQTT Clients. This format may then be consumed by the third-party endpoint and broken down in an appropriate way. All data types are fully supported by the IoT Gateway with the exception of String Arrays. Strings Arrays may only be read and not written to from the agents that support writes.

Standard Template Data Format

The standard template pushes data in a JSON format via the REST and MQTT Clients. The data structure for these agents looks like the following sample by default:

```
{"timestamp":1438011255230,
 "values":
  [
  {"id":"Channel1.Device1.Tag1","v":"250","q":true,"t":1438011254668}
  ]
}
```

The components used in the above sample are defined as follows:

- timestamp = The time, in milliseconds, that the data was published to the endpoint since the UNIX epoch
- id = The unique name of the tag in KEPServerEX
- v = The value of the tag as a string
- q = True means good quality update, false means bad (i.e. lost communications to the underlying device or invalid configuration)
- t = The time the tag data was sampled inf milliseconds since the UNIX epoch

The format of a 2 by 2 array is as follows:

```
{"timestamp":1438011255653,
 "values":
  [
  {"id":"Channel1.Device1.Tag1","v":"[1,2][3,4]","q":true,"t":1438011254924}
  ]
}
```

Note: When writing an array, all opening and closing brackets, as well as commas between the array elements, must be included. All values for an array must be written at one time. A missing value in the array prevents the entire array from being written.

The REST server uses a similar format with additional identifiers as detailed below.

For a Browse request, a JSON list of the IoT item names available with a “succeeded and “reason” field is returned. The reason field remains empty if succeeded is true.

```
{"browseResults":
  [
   {"id":"Channel1.Device1.Tag1”}
```

```
    ],
  "succeeded":true,"reason":""
}
```

For a Read request, a JSON list of the IoT item names and values with an "s" for success and "r" for reason is returned, such as:

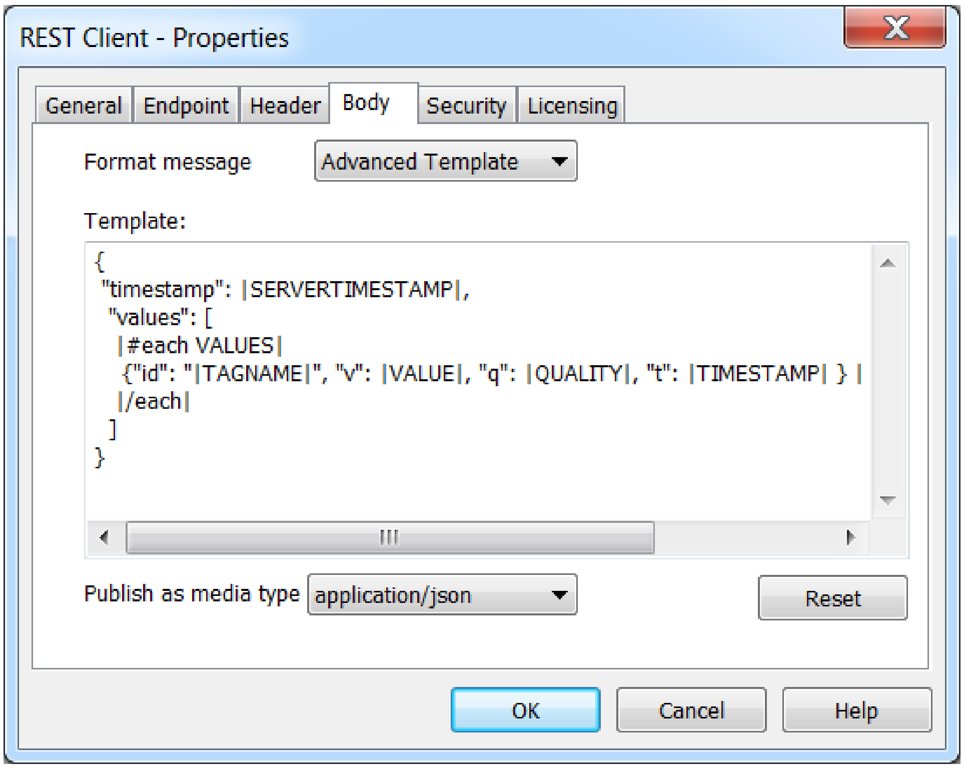
```
{"readResults":
  [
   {"id":"Channel1.Device1.Tag1","s":true,"r":"",
   "v":4878,"t":1444307548259}
  ]
}
```

For a Write request, a JSON list of the IoT items with an "s" for success and "r" for reason is returned, such as:

```
{"writeResults":
  [
   {"id":"Channel1.Device1.Tag1","s":true,"r":""}
  ]
}
```

Advanced Template Data Format

The IoT Gateway pushes data in a standard JSON format via the REST and MQTT Clients. This format may then be consumed by the third party endpoint and broken down in an appropriate way. The REST Client agent and MQTT agent have the ability to use an advanced template for pushing data. This template engine is based off of a subset Handlebars. The Advanced Template allows more complete control over the payload format. In addition to JSON, formats like XML and CSV can be generated using this template. The Advanced Template is a drop down selection under the Body or Message tab on the MQTT and REST Client agents.



Format message: Advanced Template

Template:

```
{
 "timestamp": |SERVERTIMESTAMP|,
  "values": [
   |#each VALUES|
    {"id": "|TAGNAME|", "v": |VALUE|, "q": |QUALITY|, "t": |TIMESTAMP| } |
   |/each|
  ]
}
```

Publish as media type: application/json

Publish as media type: This setting changes the content-type header of the REST client to be specific for the output of the template. This setting is not available under the MQTT Advanced template as it does not apply.

Variables

Like the Standard Template, variables can be inserted into the template representing publish time and data changes. Top-level variables like SERVERDATE and SERVERTIMESTAMP can be inserted anywhere in the template.

|SERVERTIMESTAMP|	Time of publish, represented as the number of milliseconds since January 1st, 1970, midnight
|SERVERDATE|	Date and time of publish as human-readable string

The VALUES variable represents a list of every data change in the publish. Each data change contains variables for the tag name, value, quality, and timestamp.

|TAGNAME|	The name of the tag ('Channel.Device.Tag')
|VALUE|	The value of the tag
|QUALITY|	"true" if the tag was read successfully, "false" if the tag could not be read (e.g. a connection issue)
|TIMESTAMP|	The time at which the tag was read, represented as the number of milliseconds since January 1st, 1970, midnight.

Syntax

The 'each' keyword allows text to be generated for each item update. The template inside the 'each' block is evaluated once for every item update in the publish. Depending on the format, this can be used to make each update a JSON object or a CSV row for example.

```
|#each VALUES|
  |TAGNAME|, |VALUE|, |QUALITY|, |TIMESTAMP|,
|/each|
```

In this example, each item update is formatted as line of comma-separated values, allowing import into a spreadsheet program or parsed by a CSV library:

```
Channel1.Device1.Tag1, 23, true, 1456150184825,
Channel1.Device2.Tag2, 1.79e308, true, 1456150184825,
...
```

Example Templates

Several example templates are provided for common scenarios.
If using a format other than JSON, be sure to set the Content-Type header in the **Header tab**. For XML data, 'Content-Type: text/xml' is suggested, while 'Content-Type: text/plain' is appropriate for CSV and most other formats.

XML Template

```
<updates>
|#each VALUES|
```

XML Sample Output

```
<updates>
  <update>
```

```
<update>
  <id>|TAGNAME|</id>
  <value>|VALUE|</value>
  <quality>|QUALITY|</quality>
  <timestamp>|TIMESTAMP|</timestamp>
</update>
|/each|
</updates>
```

```
    <id>Channel1.Device1.Tag1</id>
    <value>23</value>
    <quality>true</quality>
    <timestamp>1456150184825</timestamp>
  </update>
  <update>
     <id>Channel1.Device2.Tag2</id>
     <value>1.79e308</value>
  …
  </update>
  …
</updates>
```

CSV Template

```
|#each VALUES|

|TAGNAME|,|VALUE|,|QUALITY|,|TIMESTA-
MP|,
|/each|
```

CSV Sample Output

```
Channel1.Device1.Tag1, 23, true, 1456150184825,
Channel1.Device2.Tag2, 1.79e308, true,
1456150184825,
...
```

JSON, wide format

This template generates a flat JSON object with no hierarchy.

Template

```
{
|#each VALUES|
  "|TAGNAME|_value": |VALUE|,
  "|TAGNAME|_quality": |QUALITY|,
  "|TAGNAME|_timestamp":
|TIMESTAMP|,
|/each|
}
```

Sample Output

```
{
"Channel1.Device1.Tag1_value": 23,
"Channel1.Device1.Tag1_quality": true,
"Channel1.Device1.Tag1_timestamp":
1456150184825,
"Channel1.Device2.Tag2_value": 1.79e308,
"Channel1.Device2.Tag2_quality": true,
…
}
```

Additional Syntax

The default advanced template contains the following syntax:

```
|#unless @last|,|/unless|
```

This can be read as: "Unless this is the last item in the list, insert a comma."
This effectively eliminates the trailing comma for the last item in a list. While this can be omitted in most cases, it is necessary if the consumer of the payload treats trailing commas as a syntax error.

Errors

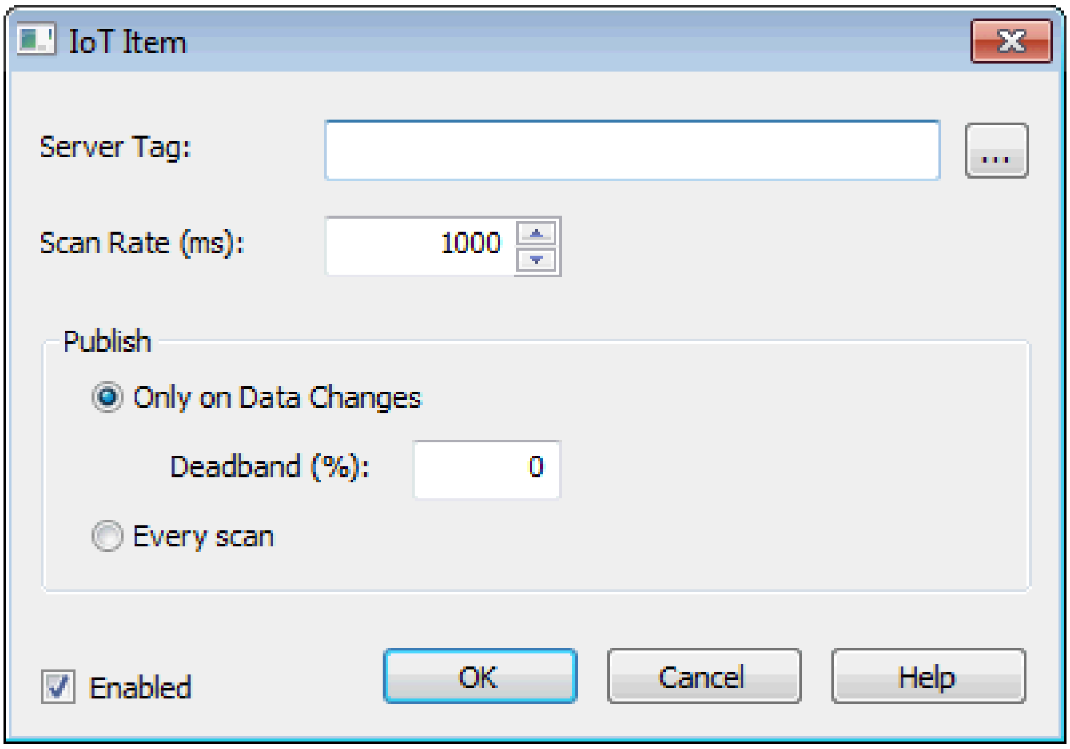
If an invalid template is defined, an error message is posted to the Event Log. This message provides the line number and cause of the error.
The agent stops publishing updates until a valid template is entered.

Adding Tags to an Agent

Once an agent is configured, it's ready to add tags. Follow the steps below to add single or multiple tags.

Adding a Single Tag

1. In the configuration window, select the agent to add the tag.
2. Right-click in the upper right pane and select **New IoT Item**.



IoT Item

Server Tag:

Scan Rate (ms): 1000

Publish

[✓ Only on Data Changes]

Deadband (%): 0

Every scan

[✓ Enabled]

Server Tag: Enter the full channel.device.name of the tag or browse to locate the single tag.

Scan Rate: the frequency, in milliseconds, at which the tag is checked for updates in value.

Only on Data Changes: sets the agent to only publish data for this tag when the value changes.
Deadband: the percentage of value change that defines the threshold of change to trigger publication. Change is based on the full range of the tag data type. A deadband of 0 means all data changes are published.

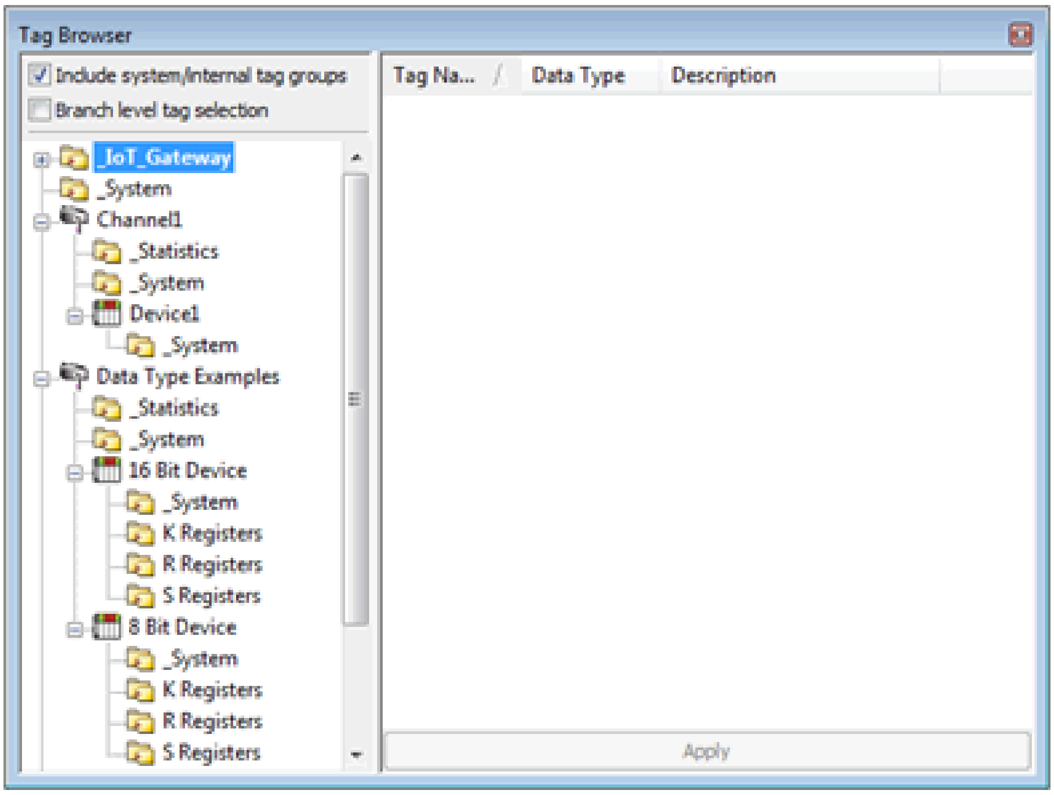
Every Scan: This forces the agent to publish data for this tag to the endpoint even if there was no change in the tag value.
Note: Tags that have a quality of "Bad" send one update with that quality and then no updates are sent until the quality returns to good.

Enabled: allows or prevents data for this tag to be published. Tags that are not enabled still count against the license count.

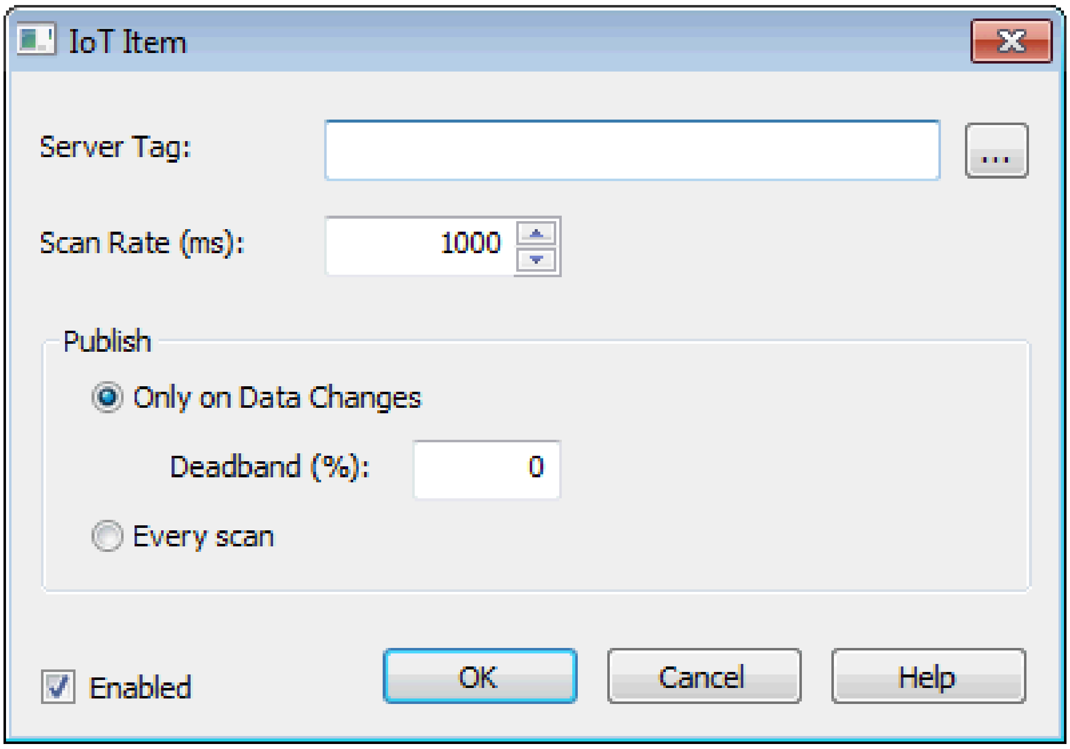
3. Click **OK**.

Adding Multiple Tags

1. In the configuration window, select the agent to which to add tags.
2. Right-click in the upper right pane and select **New IoT Items** or click on the **New IoT Items** button in the tool bar.
3. In the Tag Browser, select the tags to publish by this agent. These tags are only available on this particular agent; not on any others configured.



4. Once the tags are selected, click **Apply**.
5. In the IoT Items dialog, define the properties for the tags being added:



Server Tag:

Scan Rate (ms): 1000

Publish

[✓ Only on Data Changes]

Deadband (%): 0

Every scan

[✓] Enabled

Scan Rate: the frequency at which the tag(s) are checked for updates

Publish:

- **Only on Data Changes**: Limits the data published to value changes.
    - **Deadband (%)** the percentage of value change that defines the threshold of change to trigger publication. Change is based on the full range of the tag data type. A deadband of 0 means all data changes are published.
- **Every scan**: forces the agent to publish data from this tag to the endpoint even if there was no change in the tag value.

Enabled: allows or prevents the tag(s) to be monitored, collected, and published. Tags are enabled by default.

Note: Tags that are not enabled still consume a license count.

6. Once configured, click **OK**.
7. Verify the tags are listed in the upper right pane of the configuration window and in the target agent or broker. (Consult the Event Log for errors if no data appears.)

Note: Tags are configured to publish on only the agent where they are added. To publish the same tag to multiple end points, the tag must be added to each agent.

System Tags

The IoT Gateway exposes some status information though IoT Gateway system tags. These tags update at a five-second interval. They may be reset to zero by writing any value to them. When configured as a tag under an IoT Gateway agent, system tags count against the overall licensed number of tags. The following list contains the system tags and a brief description:

_DroppedEvents: the number of tag updates that were not successfully published due to the agent buffer being full. This can occur if the configured endpoint is not up or responding

_PublishesSent: the total number of data push events an agent has successfully made on an endpoint. Each successful publish may encompass a single or many tag updates.

Importing / Exporting CSV Files

Configuring tags for each agent may be done through the import of a comma delineated file. This is done on a per agent basis. Exporting the list of tags of an already configured agent may also be done this way. To import or export a list of tags, right click on the agent and select either “Import CSV…” or “Export CSV…”. Selecting “Export CSV…” will bring up a dialog box asking what to name and where to save the CSV file. Once saved, edit the CSV file as desired. This file may then be re-imported to this agent or any agent as long as the formatting is maintained. To start with an appropriately formatted file, it is recommended that a single tag is added to an agent. Once added export the CSV file for that agent for use as a template. This provides the format for any added tags.

Note: Importing a CSV file removes existing tags under the agent and adds what is in the CSV file. To add to the existing tags, use the export option first, add the new tags to that CSV file, and then Import that complete CSV file.

Troubleshooting

Event Log

The KEPServerEX Event log provides information pertaining to each agent connection and the status of the gateway service.

Please refer to it and the message list to resolve issues.

Data Loss

While rare under normal circumstances, errors reported in the event log can indicate dropped data. This can be due to one of the following conditions:

- Continuous incoming data rate is too fast (>100,000 updated tags per second).
- Unable to communicate with the third party endpoint and the gateway buffer is full.

The gateway buffer is configured on a per agent basis. Once the buffer maximum has been reached the gateway will drop new data coming in to the gateway for that agent. Older data is retained in the buffer until it may be pushed to the third-party endpoint.

Note: If there are any tags in an agent's buffer when a change is made to its configuration, the tags are lost.

See Also:
Event Log Messages

Event Log Messages

The following messages may be generated. Click on the link for a description of the message.

Browse rejected: no user credentials were provided in the request and anonymous requests are currently disabled.
Browse rejected: the credentials for user <user> are invalid.
Connection restored to server: <gateway>. Reinitializing server configuration.
Data change event buffer overrun; dropping updates. Ensure that the IoT Gateway service is running or reduce the volume of data collected.
Error adding item <tag>. This item already exists in connection <agent name>.
Error adding item <tag> to connection <agent name>.
Error importing CSV data. Invalid CSV header.
Error importing CSV data. No item records found in CSV file.
Error importing CSV item record <tag>. Update rate <update rate> is out of range, setting to <valid update>.
Error importing CSV item record <tag>. No update rate found, setting to <update rate>.
Error importing CSV item record <tag>. Deadband <deadband rate> is invalid. Deadband set to <valid deadband>.
Error importing CSV item record <tag>. No deadband value found, setting to <valid deadband>.
Failed to connect to server: <gateway>. Please verify this connection information is correct and that the host can be reached.

Failed to connect to server: <URL and port>. Please verify this connection information is correct and that the host can be reached.
Failed to create JVM using JRE at <path to JRE>.
Failed to import server instance cert: <agent name>. Please use the Administration utility to re-issue the certificate.
Failed to initialize the JVM: insufficient memory available (requested initial=<MB>, max. =<MB>).
Failed to initialize the JVM: JNI error <error>.
Failed to initialize the IoT Gateway.
Failed to launch IoT Gateway: no suitable 32-bit JRE was configured or found.
Failed to load agent <agent name>: invalid payload specification.
Failed to load project: <agent URL> is not a valid address.
Failed to load XML project. Item <tag> already exists in connection <agent name>.
Failed to start IoT Gateway service.
Failed to start IoT Gateway service. Please ensure arguments <Java variables> are valid.
IoT Gateway failed to start. Failed to bind to port <port>.
IoT Gatewayusing JRE at <path to JRE>.
Item <tag> on connection <agent name> is now licensed and sending data.
Missing server instance certificate <agent name>. Re-issue the certificate using the Administration utility.
MQTT agent <agent name> disconnected. Reason - Connection lost.
MQTT agent <agent name> dropped data change events.
MQTT agent <agent name> failed to parse payload.
MQTT agent <agent name> failed to parse payload template.
MQTT agent <agent name> failed to publish. Reason - <Broker URL>.
MQTT agent <agent name> failed to publish. Reason - Connection reset.
MQTT agent <agent name> failed to publish. Reason - The template is invalid.
MQTT agent <agent name> failed to publish. Reason - Unable to connect to server.
MQTT agent <agent name> is connected to broker <broker URL>.
MQTT agent <agent name> failed to process write request on topic <MQTT topic>. Reason - <JSON error>.
Property <name> is receiving incompatible data updates of type <data type> -defined as type <datatype>.
Property <name> was successfully updated and is no longer in an error state.
ThingWorx agent <name> failed to publish - reason: <reason>.
ThingWorx agent <name> connected to ThingWorx platform.
Failed to define property <name> on Thingworx agent <name>.
ThingWorx agent <name> dropped data-change events.
Read rejected for item <tag>: no user credentials were provided in the request and anonymous requests are currently disabled.
Read rejected for item <tag>: the credentials for user <user> are invalid.
Read rejected for item <tag>. The tag is disabled.
Read rejected for item <tag>. The tag has not been added to the plug-in.
REST client <agent name> dropped data change events.

Browse rejected: no user credentials were provided in the request and anonymous requests are currently disabled.

Message Type:
Security

Possible Cause:
Anonymous access is disabled, but no credentials were sent with the request.

Solution:
Enable anonymous access on the REST Server agent or enter a valid username and password.

Browse rejected: the credentials for user <user> are invalid.

Message Type:
Security

Possible Cause:

1. The credentials sent with the request are invalid or do not have browse permissions.
2. Anonymous access is disabled, but invalid credentials were sent with the request.

Solution:
Verify the username and password are correct and have adequate rights before trying the request again.

Connection restored to server: <gateway>. Reinitializing server configuration.

Message Type:
Informational

Possible Cause:
This message is logged when the plug-in reconnects to the gateway service, such as when there is a java change or the runtime is re-initialized.

Data change event buffer overrun; dropping updates. Ensure that the gateway service is running or reduce the volume of data collected.

Message Type:
Warning

Possible Cause:
The plug-in is unable to communicate with the gateway service and has started to lose data.

Solution:

1. Verify the gateway service is running.
2. Verify the gateway is not disabled in the Windows Services.
3. Verify the configured gateway port is not in use by another service.

Error adding item <tag> to connection <agent name>.

Message Type:
Error

Possible Cause:
The tag or tag name is invalid.

Solution:
1. Correct the name of the item or tag for which data is desired and re-try the request.
2. Create a new tag with the same address and a different name.

Error adding item <tag>. This item already exists in connection <agent name>.

Message Type:
Error

Possible Cause:
A tag with this name already exists under this agent.

Solution:
1. Verify the name of the item or tag for which data is desired and correct the request.
2. Create a new tag with the same address, but a different name, to import it under the same agent.

Error importing CSV data. Invalid CSV header.

Message Type:
Error

Possible Cause:
The header information or format in the CSV file import is missing data or is invalid.

Solution:
1. Export a new CSV file from an existing agent and use that as a template.
2. Correct the header information or format according to the instructions on headers.

See Also:
Importing / Exporting CSV Files

Error importing CSV data. No item records found in CSV file.

Message Type:
Error

Possible Cause:
The CSV file was not a valid format or contained no data.

Solution:

1. Export a new CSV file from an existing agent and use that as a template.
2. Correct the information or format of the CSV file.

See Also:
Importing / Exporting CSV Files

Error importing CSV item record <tag>. Update rate <update rate> is out of range, setting to <valid update>.

Message Type:
Warning

Possible Cause:
Tags in the import file included invalid update rate information.

Solution:
Verify all fields have valid data in the CSV files.

See Also:
Importing / Exporting CSV Files

Error importing CSV item record <tag>. No update rate found, setting to <update rate>.

Message Type:
Warning

Possible Cause:
Tags in the import file are missing the update rate information.

Solution:
Verify all fields have valid data in the CSV files.

See Also:
Importing / Exporting CSV Files

Error importing CSV item record <tag>. Deadband <deadband rate> is invalid. Deadband set to <valid deadband>.

Message Type:
Warning

Possible Cause:
Tags in the import file include invalid deadband information.

Solution:
Verify all fields have valid data in the CSV files.

See Also:
Importing / Exporting CSV Files

Error importing CSV item record <tag>. No deadband value found, setting to <valid deadband>.

Message Type:
Warning

Possible Cause:
Tags in the import file are missing deadband information.

Solution:
Verify all fields have valid data in the CSV files.

See Also:
Importing / Exporting CSV Files

Failed to connect to server: <gateway>. Please verify this connection information is correct and that the host can be reached.

Message Type:
Error

Possible Cause:
The server cannot communicate with the gateway service.

Solution:

1. Verify the gateway service is running.
2. Verify the configured gateway port is not in use by another service.

Failed to connect to server: <URL and port>. Please verify this connection information is correct and that the host can be reached.

Message Type:
Error

Possible Cause:
The port configured for communications between the plug-in and gateway is in use by another process.

Solution:
Change the connection port in **Administration | Settings | IoT Gateway** tab.

Failed to create JVM using JRE at <path to JRE>.

Message Type:
Error

Possible Cause:
The installed JRE is unable to create the JVM instance.

Solution:

1. Re-install the latest version of Java.
2. Verify that the gateway is set to use an appropriate JRE in the **Administration | Settings | IoT Gateway** tab.

Failed to define property <name> on ThingWorx agent <name>.

Message Type:
Warning

Possible Cause:
A tag in the IoT Gateway has a data type that is not supported by ThingWorx.

Solution:
Remove the tag or verify its settings.

Failed to import server instance cert: <agent name>. Please use the Administration utility to re-issue the certificate.

Message Type:
Security

Possible Cause:
The REST server SSL certificate could not be imported.

Solution:
Re-issue the certificate through **Administration | Settings | IoT Gateway | Manage Certificates**.

Failed to initialize the JVM: insufficient memory available (requested initial=<MB>, max. =<MB>).

Message Type:
Error

Possible Cause:
The computer has insufficient memory to start the JVM.

Solution:
The initial and maximum memory levels in the **Administration tool | Settings | IoT Gateway | Advanced** settings should be removed.

Failed to initialize the JVM: JNI error <error>.

Message Type:
Error

Possible Cause:
There is an issue with the Java JRE.

Solution:

1. Reinstall a valid 32-bit Java JRE version 7 or higher.
2. Verify that the gateway is set to use an appropriate JRE in the **Administration tool | Settings | IoT Gateway** tab.

Failed to initialize the IoT Gateway.

Message Type:
Error

Possible Cause:
The IoT Gateway is unable to start.

Solution:
Re-run the installation and choose to repair the setup.

Failed to launch IoT Gateway: no suitable 32-bit JRE was configured or found.

Message Type:
Error

Possible Cause:
A valid 32-bit Java JRE version 7 or higher was not found on the computer.

Solution:

1. Re-install the latest version of Java.
2. Verify that the gateway is set to use an appropriate JRE in the **Administration | Settings | IoT Gateway** tab.

Failed to load agent <agent name>: invalid payload specification.

Message Type:
Error

Possible Cause:

1. The JSON payload contains invalid or disallowed content.
2. Invalid advanced template settings during an XML project import.

Solution:

1. Adjust the payload section of the XML to be a valid data section with correct name-value pairs.
2. Correct the template and re-import the XML project file.

Failed to load project: <agent URL> is not a valid address.

Message Type:
Error

Possible Cause:
The agent specified has an invalid URL format.

Solution:
Correct the URL to a valid format and try importing the file again.

Failed to load XML project. Item <tag> already exists in connection <agent name>.

Message Type:
Warning

Possible Cause:
There is a duplicate tag under an agent in the XML project.

Solution:
Remove or correct the duplicate tag in the XML before attempting to import again.

Failed to start IoT Gateway service.

Message Type:
Error

Possible Cause:
The gateway service is set to **Manual** or there is no appropriate Java JRE installed.

Solution:

1. Under Windows Services, verify that the IoT Gateway service is set to **Manual.**
2. Verify that a valid 32-bit Java JRE version 7 or higher is installed.
3. Install a new version of Java that meets the system requirements.

Failed to start IoT Gateway service. Please ensure arguments <Java variables> are valid.

Message Type:
Error

Possible Cause:
Invalid JRE arguments were added to the advanced settings.

Solution:
Remove or correct any advanced settings from **Administration | Settings | IoT Gateway | Advanced Settings**.

IoT Gateway using JRE at <path to JRE>.

Message Type:
Informational

Possible Cause:
This message appears when the gateway starts, indicating the JRE being used.

IoT Gateway failed to start. Failed to bind to port <port>.

Message Type:
Error

Possible Cause:
The gateway service was unable to use the port assigned in the Administration tool.

Solution:
Change the port in **Administration | Settings | IoT Gateway** tab to an available unused port.

Item <tag> on connection <agent name> is now licensed and sending data.

Message Type:
Informational

Possible Cause:
The referenced tag was disabled due to license limitation, but is now sending data.

Missing server instance certificate <agent name>. Re-issue the certificate using the Administration utility.

Message Type:
Security

Possible Cause:
The REST server SSL certificate is missing or invalid.

Solution:
Re-issue the certificate through **Administration | Settings | IoT Gateway | Manage Certificates**.

MQTT agent <agent name> disconnected. Reason - Connection lost.

Message Type:
Error

Possible Cause:
The broker is unreachable.

Solution:

1. Verify that the broker is online and the network connection is functioning properly.
2. Check the configured URL for the agent and verify a properly configured broker exists at that address.
3. Verify that communication is not being blocked by a hardware or software firewall or filter.

MQTT agent <agent name> dropped data change events.

Message Type:
Warning

Possible Cause:
The broker is unreachable.

Solution:

1. Verify that the broker is online and the network connection is functioning properly.
2. Check the configured URL for the agent and verify a properly configured broker exists at that address.
3. Verify that communication is not being blocked by a hardware or software firewall or filter.

MQTT agent <agent name> failed to parse payload.

Message Type:
Error

Possible Cause:
The JSON payload has invalid or disallowed content in it.

Solution:
Adjust the Format and Expansion of |VALUES| boxes on the Message tab to remove the incorrect information.

MQTT agent <agent name> failed to parse payload template.

Message Type:
Error

Possible Cause:
The formatting of the advanced template is incorrect or missing characters.

Solution:
Verify no characters are missing and that the template logic is valid. Correct any issues.

See Also:
Advanced Template Data Format

MQTT agent <agent name> failed to process write request on topic <MQTT topic>. Reason - <JSON error>.

Message Type:
Error

Possible Cause:
The JSON payload has invalid or dis-allowed content in it.

Solution:
Adjust the JSON payload to match the expected format as described under MQTT Subscriptions.

MQTT agent <agent name> failed to publish. Reason - <broker URL>.

Message Type:

Error

Possible Cause:
The broker is unreachable.

Solution:
1. Verify that the broker is online and the network connection is functioning properly.
2. Check the configured URL for the agent and verify a properly configured broker exists at that address.
3. Verify that communication is not being blocked by a hardware or software firewall or filter.

MQTT agent <agent name> failed to publish. Reason - Connection reset.

Message Type:
Error

Possible Cause:
The agent is configured for an SSL connection and the broker does not support SSL or is not configured for SSL connections.

Solution:
1. Check the URL and port that the agent is using and verify that it is an SSL-enabled endpoint.
2. Change the agent URL to use TCP rather than SSL and try again.

MQTT agent <agent name> failed to publish. Reason - Unable to connect to server.

Message Type:
Error

Possible Cause:
No valid MQTT broker at the URL provided or communication is blocked.

Solution:
1. Verify that the broker is online and the network connection is functioning properly.
2. Check the configured URL for the agent and verify a properly configured broker exists at that address.
3. Verify that communication is not being blocked by a hardware or software firewall or filter.

MQTT agent <agent name> is connected to broker <broker URL>.

Message Type:
Informational

Possible Cause:
This message is posted once a successful connection is made with an MQTT broker.

Solution:
Proceed.

MQTT agent <agent name> publish failed. Reason - The template is invalid.

Message Type:
Error

Possible Cause:
The formatting of the advanced template is incorrect or missing characters.

Solution:
Verify no characters are missing and that the template logic is valid. Correct the issues.

See Also:
Advanced Template Data Format

Property <name> is receiving incompatible data updates of type <data type> -defined as type <data type>.

Message Type:
Error

Possible Cause:
The data type of a tag in the IoT Gateway does not match the type setting for the ThingWorx thing.

Solution:
Correct the data type in the ThingWorx composer to match the type of the tag sent from the IoT Gateway.

Property <name> was successfully updated and is no longer in an error state.

Message Type:
Information

Possible Cause:
The data type now matches the ThingWorx platform.

Solution:
Proceed.

Read rejected for item <tag>: no user credentials were provided in the request and anonymous requests are currently disabled.

Message Type:
Security

Possible Cause:
Anonymous access is disabled or no credentials were sent from the client.

Solution:

1. Enable anonymous access on the REST server agent.
2. Send valid credentials in the authentication section of GET or POST.

Read rejected for item <tag>: the credentials for user <user> are invalid.

Message Type:
Security

Possible Cause:

1. The credentials sent with the request are invalid or do not have read permissions.
2. Anonymous access is disabled, but invalid credentials were sent with the request.

Solution:
Verify the username and password are correct and have adequate rights before trying the request again.

Read rejected for item <tag>. The tag is disabled.

Message Type:
Error

Possible Cause:
The tag has been added under the IoT Gateway agent, but is disabled.

Solution:

1. Locate the tag under the agent.
2. Right-click on the tag and select **Enable** from the menu.

Read rejected for item <tag>. The tag has not been added to the plug-in.

Message Type:
Error

Possible Cause:
The tag has not been added to the IoT Gateway.

Solution:
Follow the steps under Adding Tags to an Agent.

REST client <agent name> dropped data change events.

Message Type:
Warning

Possible Cause:
The REST server is unreachable.

Solution:

1. Verify that the REST server endpoint is online and the network connection is functioning properly.
2. Check the configured URL for the agent and verify a properly configured REST server exists at that address.
3. Verify that communication is not being blocked by a hardware or software firewall or filter.

REST client <agent name> failed to parse payload.

Message Type:
Error

Possible Cause:
The JSON payload has invalid or disallowed content in it.

Solution:
Adjust the Format and Expansion of |VALUES| boxes on the Message tab to remove the incorrect information.

REST client <agent name> failed to parse payload template.

Message Type:
Security

Possible Cause:
The formatting of the advanced template is incorrect or missing characters.

Solution:
Verify no characters are missing and that the template logic is valid. Correct the issues.

REST client <agent name> processing update.

Message Type:
Informational

Possible Cause:
This message is posted when a change is made to the REST client configuration.

Solution:
Proceed.

REST client <agent name> publish failed. Reason - Connection refused: connect.

Message Type:
Error

Possible Cause:
A valid REST server endpoint has gone offline.

Solution:

1. Verify that the REST server endpoint is online and the network connection is functioning properly.
2. Check the configured URL for the agent and verify a properly configured REST server exists at that address.
3. Verify that communication is not being blocked by a hardware or software firewall or filter.

REST client <agent name> publish failed. Reason - Read timed out.

Message Type:
Error

Possible Cause:
Publishing to a HTTP endpoint with HTTPS enabled.

Solution:
Edit the agent endpoint URL to use HTTP rather than HTTPS and try again.

REST client <agent name> publish failed. Reason - SSL configuration error.

Message Type:
Error

Possible Cause:
The client certificate has not been imported into the Microsoft computer-level root trusted certificate store.

Solution:
Import the proper client certificate into the certificate store. These instructions are listed in the **Configuring a Self-Signed Certificate** section.

REST client <agent name> publish failed. Reason - The template is invalid.

Message Type:
Security

Possible Cause:
The formatting of the advanced template is incorrect or missing characters.

Solution:

Verify no characters are missing and that the template logic is valid. Correct the issues.

See Also:
Advanced Template Data Format

REST client <agent name> publish failed. Reason - Unexpected EOF.

Message Type:
Error

Possible Cause:
Publishing to a HTTPS endpoint without HTTPS enabled in the URL.

Solution:
Edit the agent endpoint URL to use HTTPS rather than HTTP and try again.

REST client <agent name> returned HTTP error <HTTP error>, buffering records.

Message Type:
Warning

Possible Cause:
The configured endpoint URL is incorrect or not accepting connections. The gateway buffers data until the endpoint comes online or is corrected in the configuration.

Solution:
The HTTP error returned from the endpoint should indicate how to establish a connection.

Examples:
A 404 error indicates the URL is incorrect.
A 401 error indicates the username or password is incorrect.

REST client <agent name> started publishing to <REST server URL>.

Message Type:
Informational

Possible Cause:
This message is posted once a successful publish is made with the configured REST server.

Solution:
Proceed.

REST server <agent name> started at <URL and port>.

Message Type:
Informational

Possible Cause:

This message is posted when the REST server is activated on the gateway.

Solution:
Proceed.

REST server <agent name> - failed to start on <URL and port>. Reason - Address already in use: bind.

Message Type:
Error

Possible Cause:
An existing REST server agent is already using this port or another service on the computer is using this port.

Solution:
Edit the port setting in the REST server agent properties to an available port.

Running with Java <full Java version>.

Message Type:
Informational

Possible Cause:
This message appears when the gateway starts, indicating the full version of Java being used.

Template error on line <number>: found: <string>.

Message Type:
Warning

Possible Cause:
The formatting of the advanced template is incorrect or missing characters.

Solution:
Verify no characters are missing and that the template logic is valid. Correct the issues.

See Also:
Advanced Template Data Format

The REST server certificate has been reissued.

Message Type:
Security

Possible Cause:
A new REST server certificate has been successfully issued from the certificate manager.

The REST server certificate has been imported.

Message Type:
Security

Possible Cause:
A new REST server certificate has been successfully imported.

Solution:
Proceed.

The REST server certificate has expired. Please use the Administration utility to re-issue the certificate.

Message Type:
Security

Possible Cause:
The current REST server SSL certificate is expired.

Solution:
Re-issue the certificate from **Administration | Settings | IoT Gateway | Manage Certificates**.

ThingWorx agent <name> connected to ThingWorx platform.

Message Type:
Information

Possible Cause:
A successful connection was made to the ThingWorx endpoint.

Solution:
Proceed.

ThingWorx agent <name> dropped data-change events.

Message Type:
Error

Possible Cause:
The ThingWorx endpoint is unreachable or responding slowly.

Solution:

1. Verify that the ThingWorx endpoint is online and the network connection is functioning properly.
2. Check the configured URL for the agent and verify a properly configured ThingWorx endpoint exists at that address.
3. Verify that communication is not being blocked by a hardware or software firewall or filter.

ThingWorx agent <name> failed to publish - reason: <reason>.

Message Type:
Error

Possible Cause:
Unable to connect to the ThingWorx endpoint.

Solution:

1. Verify that the ThingWorx endpoint is online and the network connection is functioning properly.
2. Check the configured URL for the agent and verify a properly configured ThingWorx endpoint exists at that address.
3. Verify that communication is not being blocked by a hardware or software firewall or filter.

Unable to send data for item <tag> on connection <agent name>. The licensed item count of <license count> items has been reached.

Message Type:
Warning

Possible Cause:
More tags are configured than the license allows.

Solution:
Remove unused tags from any configured agents or apply a license that allows for more tags.

Unable to start secure REST server <agent name> at <URL and port>: missing or invalid certificate.

Message Type:
Error

Possible Cause:
The SSL certificate is missing is or invalid.

Solution:

1. Verify that certificate exists in the product path at: C:\ProgramData\Kepware\KEPServerEX\V5\IoT Gateway\RESTServer\cert
2. Re-issue the REST server certificate from **Administration | Settings | IoT Gateway | Manage Certificates**.

Unable to use network adapter <network adapter> for REST server <agent name>. Binding to localhost only.

Message Type:
Warning

Possible Cause:
The network adapter in the project does not match any found on the current machine.

Solution:
Use the Endpoint tab of the REST server to adjust the network adapter.

Unsupported JVM: please install or configure a 32-bit Java 1.7 or higher JRE or JDK.

Message Type:
Error

Possible Cause:
There is no valid version of Java installed for the gateway.

Solution:
Install a valid 32-bit Java JRE version 7 or higher.

Write request failed on item <tag>. The write data type <data type> cannot be converted to the tag data type <data type>.

Message Type:
Warning

Possible Cause:
The write payload was of a data type that cannot be written to the selected tag.

Solution:
Verify that the tag data type being written is correct and that the data being written matches acceptable values for that data type.

Write rejected for item <tag>; invalid write format.

Message Type:
Error

Possible Cause:
The data parsed is missing information or formatting.

Solution:
Verify that the data being written is in a valid JSON format and matches the examples in the MQTT and REST server sections of this document.

See Also:
MQTT Client Message

REST Client Body

Write rejected for item <tag>: no user credentials were provided in the request and anonymous requests are currently disabled.

Message Type:
Security

Possible Cause:
Anonymous access is disabled, so credentials must be provided, but none were sent from the client.

Solution:
Enable anonymous access on the REST server agent or enter a valid username and password.

Write rejected for item <tag>: the credentials for user <user> are invalid.

Message Type:
Security

Possible Cause:

1. The credentials sent with the request are invalid or do not have write permissions.
2. Anonymous access is disabled, but invalid credentials were sent with the request.

Solution:
Verify the username and password are correct and have adequate rights before trying the request again.

Write rejected for item <tag>; unsupported data type <type>.

Message Type:
Error

Possible Cause:
The tag data type does not match the data to be written.

Solution:

1. Verify it is not a string tag as the target to write. Strings are not supported.
2. Verify that the value is within the limits of the data type of the tag.

Write rejected for item <tag>. The tag is disabled.

Message Type:
Error

Possible Cause:
The tag has been added under the IoT Gateway agent, but is disabled.

Solution:

1. Locate the tag under the agent.
2. Right-click on the tag and select **Enable** from the menu.

Write rejected for item <tag>. The tag has not been added to the plug-in.

Message Type:
Error

Possible Cause:
The tag has not been added to the IoT Gateway.

Solution:
Follow the steps under **Adding Tags to an Agent**.

Index