English

Asema IoT Edge

User Manual

ASEMA

Table of Contents

1. Introduction

Asema IoT Edge is a compact software for gateways that offers interfacing for sensors, offline logging, logic functionality and can operate autonomously with fast response time. In the internet-of-things network, instances of Asema IoT Edge and Asema IoT Central connect together and share data, objects and controls.

With Asema IoT Edge, you can:

- Set up IoT objects that feed data in the IoT system.
- Create automation rules to control the objects connected to the system.
- Configure user access to the objects.

Basically, Asema IoT Edge is basically a stripped-down version of Asema IoT Central, so part of the functionality is missing as it is not needed for receiving and processing the data and controlling objects on the spot.

In order to function, Asema IoT Edge instances must be connected to Asema IoT Central. All objects added to the system are pooled together, so you can easily operate them from Asema IoT Central as well.

2. Installation and startup

To install Asema IoT Edge, obtain the corresponding installation package and start the installer as described in the following sections.

> **Important**
>
> Remember to install the package that fits the type of your operating system. If you are running a 32-bit system, you need the 32-bit installer. A 64-bit operating system can run both 32-bit and 64-bit software.

2.1. Raspberry Pi

Asema IoT Edge for Raspoberry Pi is available as an RedHat Package Manager (rpm) package. Download the package, place it on your Raspberry Pi and then issue:

```
rpm -Uhv [package]
```

Replace [package] with the path and name of the package you downloaded.

2.2. Linux

Asema IoT Edge for Linux uses the RedHat Package Manager (rpm) packages. Depending on the Linux distribution you use, installing it differs. In most distributions it is enough to double-click on the RPM package and then follow instructions on screen.

If you need to install the rpm from command line, execute the following command as root:

```
rpm -Uhv asema_iot_edge_for_linux.rpm
```

2.3. Running Asema IoT Edge

You can run Asema IoT Edge either with a user interface or as headless. The core functionality in both modes is the same. On a device with no display or a system where the default interface of Asema IoT Edge does not fit the display or the display is not recognized correctly, the headless mode is the natural choice.

To start the application with user interface, simply run the program binary that was installed. For example, if the program binary is under /usr/local/bin, run it as `/usr/local/bin/asema_iot_edge`. To start it in headless mode, add the -h flag. So, for example, `/usr/local/bin/asema_iot_edge -h`.

2.4. Obtaining a license

Asema IoT Edge should always be run together with a central system, namely one or several Asema IoT Central instances, as the purpose of an edge device is to offer a local proxy for data collection and control for the central system(s).

Consequently, the Asema IoT Edge licensing is bound to the license of the Asema IoT Central (the "host"). Asema IoT Edge itself has no individual license. Rather it checks from the host whether a license is available there and whether this edge instance fits the licensing policy and possible limits.

For the Asema IoT Edge license validation to work, two things need to be in place when the Asema IoT Edge instance starts up:

- At least one Asema IoT Central instance with free Edge license slots must be online and accessible by the Asema IoT Edge instance.

- The IP address, domain address or CloudRoute address of the Asema IoT Central instance must be entered as the host system into the Asema IoT Edge system settings.

Once these two are in place, license validation happens automatically.

2.5. Logging into the admin interface

Once the software has started, you can login to the system. Bu default, it runs on port 8080 on your edge device. To login, open a browser and enter the following URL: http://[ip or domain of your device]:8080. So, for example, to access the system on 192.168.1.10, type http://192.168.1.10:8080.

The default user is `admin` and the default password is `admin`

Important

Change the default admin password in the Users menu as soon as you log in to the system to avoid security breaches.

3. System settings

System settings can be found under the System menu. Most settings related to configuring and maintaining the system and setting up the services are located here.

3.1. Host system

Asema IoT Edge must be connected to a host server running Asema IoT Central. When Asema IoT Edge is started, the system checks the software licence from the host system. So, to make your Asema IoT Edge work, type in the host system address, test and save it.

> **Important**
>
> A free demo licence lets you connect only one Asema IoT Egde to the host system. To connect more Asema IoT Edge instances, obtain a paid Asema IoT Central licence.

3.2. Paths and identifiers

When you manage a large system and need to install components, take backups, connect to APIs, an so on, it is easy to forget where exactly the necessary files are and how the network was configured. Worry not, Asema IoT instances summarize all of them for you. In the System > Paths and identifiers section, you can find all URLs and locations you need for configuring the system and using APIs.

3.3. Basic details

Basic details of your system are needed to access Smart API (i.e. sharing functionality) and display information in the Smart API Find registry where other users search for offers.

3.4. Notification listener

As a notification listener, Asema IoT listens to an MQTT topic and passes on that data to some object for processing. The topic can either be assigned to a particular object or you can have Asema IoT create an object on the fly as it receives a message.

To set up your Asema IoT application as a listener, go to System > Notification listener. Then add a topic by specifying:

- Name of the topic.
- Publisher ID, Publisher type, Conversation ID, Conversation type make up the topic path. You can replace any part of this path with the hash (#) wildcard character.
- URL encode. You may choose to URL encode the topic to handle for instance forward slashes within the levels. By default the four parts of the topic are handled as four levels separated by a forward slash as is common in MQTT i.e. your topic will look like this: publisherID/publisherType/conversationID/conversationType. However, if any of these levels itself uses a forward slash, this will become a problem. You may for instance in Smart API applications need to use a Publisher ID that is a URI i.e. a string that starts with http://. This causes trouble as the forward slashes will mark it as two additional levels. Tick the URL encode option to encode each level so that forward slashed become %2F and pass through the broker without confusion.
- Bind to schema that should be used for parsing the messages.

· Autocreate an object for receiving the messages. It will appear in the list of objects.

Tick this box if you want to create an object automatically. Select a schema to bind it to. Without a schema, Asema IoT doesn't know how to process the data contained in the MQTT message.

> **Note**
>
> The schemas available in this menu are defined under Objects > Server API objects > API Schemas.

Leave the box unticked to assign the topic to a particular object. Define the topic here, then go to Objects > API objects and create or select an object. Choose MQTT as a protocol and then select the topic you've just created from the dropdown menu. Note that if you assign the topic to an object, the selection of the schema in system settings under Notification listener has no effect. The schema selected for the object in API objects is used instead.

3.5. Notification sender

Asema IoT can publish notifications about objects and about system events (system errors and warnings and changes in the geographic position of the system).

To set up your Asema IoT application as a publisher of system events, go to System > Notification sender. Add a broker server and enter topic names to send messages to.

To configure publishing notifications about objects, go to System > Notification sender and specify the broker server. Then go to Objects > Configure > Notifier and enter the notification settings there.

3.6. CloudRoute VPN

CloudRoute VPN is a VPN solution that connects CloudRoute compatible systems together over an encrypted Virtual Private Network. It secures traffic and makes it possible to directly communicate between systems even when they sit in separate private networks behind a firewall. CouldRoute VPN access is group-based. All systems that belong to the same group see each other and can communicate with each other. Access to a group is granted with a username and a password. Note that the username and password only give access to the VPN tunnel i.e. it establishes a connection to a remote system but does not log in. A system in a group may require further login credentials to access its data.

CloudRoute can also be used to access the web admin interface of Asema IoT remotely. To do so, the admin must be enabled in Asema IoT and you need to login to asema.com management service with the username and password of the group.

Asema IoT can also act as a port forwarder service to some other known service (such as a database or remote shell) that resides on the same machine as Asema IoT. This can help, for instance, in remotely managing a full software infrastructure needed by an IoT application.

To access Asema IoT remotely via VPN, go to System > CloudRoute VPN and choose Enable CloudRoute VPN for remote admin.

To use your Asema IoT application as a VPN gateway to access other servers, go to System > CloudRoute VPN and enter comma-separated ports.

3.7. Webserver

The web (HTTP) server is the main application protocol server Asema IoT uses for the web admin interface and the majority of the APIs of the system. To set up a web server for running Asema IoT, go to System > Webserver settings.

In the Webserver settings, you can configure where the files served by the webserver (most notably the Web applications) will be stored. Additionally you can set up the HTTP URL paths for three functions: public pages, server side scripts, and proxying.

- Public pages. The address path determines from which path pages and web applications are available. For example, if the path is set to `mypath`, then a file "mypage.html" stored in the public pages filesystem path would be available at http://[asema iot address]/mypath/mypage.html. Simularly the webapp called "myapp" is at http://[asema iot address]/mypath/myapp/
- Server side scripts. This is the path for accessing server side script logic. If you would program a script and set its request path to "myscript" and then set the Server side scripts base path to "applicationscripts", your script would be accessed at http://[asema iot address]/applicationscripts/myscript
- Proxied calls. Asema IoT Edgecan act as a simple HTTP proxy and will relay the calls given to it to some external server. This feature is especially useful if you are making a web application that needs other web application servers such as node.js to run but do not want to install a separate proxy server. The address of the other server (server to be proxied to) is configured by setting it to Default proxy target server. So for example, if you set the procy target server to http://192.168.0.100:8080and the Proxied calls base path to "myproxy", all calls to http://[asema iot address]/myproxy/mypath/will be forwarded to http://192.168.0.100:8080/mypath.

> **Note**
>
> Note that the ports used by the HTTP server are configured in the Network servers section of the setup.

3.8. Network servers

To specify network servers for connecting to other Asema IoT systems, go to System > Network servers. Define the following settings:

Require valid credentials to access APIs — Whether APIs can be accessed without API key. If on, this setting overrides the options chosen in Sharing > Access control.

HTTP, CoAP, MQTT, WebSocket server ports — Ports for accessing your Asema IoT from inside the firewall.

HTTP, CoAP, MQTT, WebSocket NATted ports — Ports for accessing your Asema IoT application from outside the firewall.

3.9. Bluetooth

Asema IoT has in-built support for short range radio protocols such as Bluetooth or NFC. With these you can connect a compatible radio device directly into a PC running Asema IoT or have two instances of Asema IoT talk to each other wirelessly over Bluetooth.

To configure Bluetooth communication, you need to define a hardware object of Bluetooth type. The object will automatically try to connect over the Bluetooth adapter on that PC.

For this to happen, you need to tell the software which Bluetooth adapter to use as you may have several of them on the PC. To do so, go to System > Bluetooth and enter the MAC address of the Bluetooth adapter to use.

3.10. Credentials

Credentials are access tokens to some system, either IoT Central itself or some system IoT Central connects to. A typical credential is a username-password pair.

There are three types of credentials in the Asema IoT:

- System credentials. System credentials limit access to the Asema IoT instance the credential is in. This type of credential gives access **to** your Asema IoT. The system credentials are API keys that must be added to the HTTP calls that are sent to the Asema IoT when accessing it using the JSON API.
- Client credentials. This type of credential opens access **from** your Asema IoT. Note that the calls to other systems are done by Network client objects. To use these credentials, they need to be added to the schemas of those objects. For more details on how to write such schemas, please see 4.3.2.3, “Network client objects”. Note that client credentials can also be written as such directly into those schemas. However, this will expose the username and password to anyone who edits the schemas. To hide such details, put the credentials into system settings. They can then be deferred to in the schemas with their IDs.
- Certificates.

3.11. Memberships

Memberships list your memberships in CloudRoute VPN groups and lets you create new groups. For more information on CloudRoute, please see 3.6, “CloudRoute VPN”

3.12. Location

The location of your Asema IoT system is used in various features of the system, such as the location stamped on recorded database values and as a starting view of all the maps where you make edits.

The latitude and longitude (WGS-84) is used for this purpose. The postal address is an additional meta information that can be used for instance when managing several installations.

The latitude and longitude information can be entered as fixed values or you may set up the system so that it tracks values from a positioning device or an object in the system.

To set the geographic location of your Asema IoT, go to System > Location. You can use the following options:

- Enter the system's postal address or fixed coordinates.
- Read the system's coordinates from a source:

  A positioning sensor on this device.

  The position of some object that has a GPS device.

  A positioning plugin (available plugins are listed at the bottom of the page).

3.13. Logging and console

To configure the logging feature of the system or view the latest log in a console, go to System > Logging and console.

In this section of the setup, you can perform the following actions:

- Restart the application.

  Click the Reload button to restart the system. Restart will empty all caches, disconnect all connections and stop all timers and other logic. These are then reloaded into memory and connections are re-established. Users will remain logged in during this process but any network connections made by peer systems must be reconnected.

- Configure system logging.

  Here you can select the verbosity of the log from very brief to very verbose. By default the log output is shown on screen i.e. the terminal window from where the system has been started if a terminal has been used. Alternatively on systems that support a journal (systemd), the log may be redirected to it. As a final alternative, you can output the log into a file which will reside in the home directory of the application. The log is size limited and will be rotated once the limit is reached.

3.14. Startup

The Startup section provides the following startup settings:

- Password and welcome screen requirement for the Asema IoT desktop application.
- Fallback settings to use if there are some system configuration errors that make the application crush:

  Automatically load rules at startup — Untick it to start the application ignoring the standard rules (for example, if there is a loop in the rule)

  Instantiate QML of custom rules at startup — Untick it to start the application ignoring the custom rules.

  Activate hardware objects at startup — Untick it to prevent loading the hardware drivers.

  Block headless startup if peer connections fail — Untick it to quit startup in case some peers are offline.

3.15. About and license

About and license section displays the software version and license information. If Asema IoT Edge failed to contact the host system, the license status is "invalid".

4. Object connectivity

4.1. Introduction

In Asema IoT, objects represent all devices — either physical or virtual — and other data sources. Setting up and changing the objects is the core task in making IoT work with the system. The setup of each object defines precisely how the device or other system is connected to Asema IoT and how it behaves. Each object comprises:

- Name. Quite simply the name of the particular object.
- Identifiers. The identification of the object that allows applications and APIs find the object in an exact fashion. See below for details on identifiers.
- Metadata. Object type-related or object-specific fields that describe what the object is like.
- Schema. A schema is a technical definition of how an object type actually operates and interfaces with the world.
- Connectivity settings. Object specific technical setup for its connectivity. The schema, i.e. the definition of an object class, and the settings, i.e. the definition of a particular object instance, together form the settings that make the object connect.

While the above defines the object, three types of continuously changing and recorded data are attached to the objects to record its actions and make it operate:

- Properties. Properties are measurable and controllable attributes of an object, for example speed, pressure, temperature and similar freely defined variables that are written into a schema. As typical objects in an IoT system are some sort of physical objects, two multidimensional properties are available to all objects irrespective of their schema: their position (latitude, longitude and altitude) and orientation (yaw, pitch and tilt).
- Events. These are things that have happened to the object, for instance if it has been broken and repaired.
- Tasks. These are things that should be done to the object, for instance do a maintenance checkup.

Each object has the following identifiers that can be used in various applications for pointing exactly to that object:

- Global Identifier (GID). Mandatory, fixed. A GID is the basic identification number every object has. When an object is created the first time, it is assigned a random identifier that is guaranteed to be unique across all systems. Once the GID is generated, you cannot change it. As the GID is mandatory, you can always find an object if you know its GID.
- Object Identifier (Smart API identifier). Mandatory, editable. The Object Identifier is an extension to the GID used by the Smart API standard and the Asema IoT sharing functionality. Just like the GID, it is assigned to an object when the object is created. It comprises a domain part (http://[domain]) and an identifier part ([random string]) where the identifier part by default is the GID. The domain part makes using Object Identifiers easier in applications that stretch across organizations. Like the GID, the Object Identifier is unique for each object. However, unlike the GID, it can be edited and changed. The Object Identifier also makes it possible to track which objects are actually acting as copies of each other. When an object is shared and resides in two different systems, the copies of the object will have different GIDs but the same Object Identifiers.
- Component Identifier. Optional, editable. The Component Identifier is an optional identifier to be used by applications to recognize an object meant for the same purpose although it may not be the same object. This identifier is used in situations where the same application runs on two different systems

but with different data. For instance, there is an application for managing a baby monitor at a house. The same application can be run by different people in different houses and they all want to monitor their particular baby monitor. To avoid configuring the unique GID of the particular baby monitor to each application instance, the application developer can specify a Component Identifier, say "ourBaby-Monitor" in the application and this is the only identifier that needs to be configured to the individual monitors. When you export and object and transfer it to another system via the import feature, the imported object will have a unique GID and a unique Object Identifier but the Component Identifier will be the same.

4.2. Object types

There are four broad types of objects in Asema IoT: Server API objects, Client API objects, Database objects and Hardware objects. As the core purpose of the system is to collect and unify data originating from various sources, all of these objects offer the same application capabilities and application API. Therefore, from an application perspective, it does not matter which object type is chosen.

However, from a connectivity point of view, the choice of object type makes all the difference as the purpose of different object types is to define how exactly the object acquires its data. So, in essence all object types **serve** their data the same way but **acquire** the data differently.

> **Note**
>
> All objects of any type are always available through both the JSON API and Smart API. If you intend to do system integration using either of these two APIs and do not require additional data source functionality for a particular object, it does not matter which object type you choose. Simply pick one that is easy to manage for you. In most cases the recommended choice is the Generic Object as it is easiest to configure and does not try to connect to any resource.

Depending on the type of connectivity, you should choose the object type using the following rules:

- Generic object. Choose this type if you do not need any object connectivity but instead need a simple object for storing data and using that data in your application logic. A generic object has no interface, just API access.
- Server API object. Choose this type when you want Asema IoT to act as a server that waits and receives the data. Typically in this case you are dealing with some other system that is able to push data into a given URL but you do not want or cannot change the format of the data that the system is pushing forward. A Server API object has a listener interface with a parser that can interpret the incoming data.
- Network client object. Choose this type when you want Asema IoT to act as a client that actively fetches the data. Typically in this case you are dealing with a network server with a public API that responds to a data query. A Network client object has a requester interface that is designed to actively interact with an external server and parsers to interpret the data.
- Database object. Choose this type if you have a database server with no network API and you'd like to fetch data from the database using SQL queries. A Database object has a database connection interface for directly talking to the database.
- Hardware object. Choose this type if you have a hardware device which will interface directly with the unit where you installed Asema IoT. A Hardware object has a hardware driver interface that can interact with the hardware in case.

Each object type has its own connectivity settings. The chapters below describe these in detail. In addition to the connectivity settings, you need a schema. The schema defines what the data you manipulate actually looks like and how it can be turned into properties of the Asema IoT object. Schemas are

shared across objects as most objects have similar data. So for instance, if a remote controlled crane sends data in a particular format and your IoT system manages one hundred cranes, there is no need to write one hundred schemas. Instead, you write one schema for a crane and assign that schema to all the cranes in the system.

Schemas control the so called parsers and serializers. A parser takes external data and parses it into the format needed by Asema IoT. A serializer does exactly the opposite: it takes data from Asema IoT and formats it into a format needed by some external system.

The format of the definitions of parsers and serializers will differ depending on the object type. As this is a broad topic, there is a separate manual on the exact details of schemas. Please refer to Object data parsing and serializing manual for details.

4.3. Creating an object

To create an object, you need to choose the object type, specify the connection parameters, select the object schema or a plugin to read the data and configure access rights.

These steps can be performed either with an object form in the Objects menu or with the object creation wizard available on the front page of the Objects setup (Create a new object button).

4.3.1. Using the object wizard

If you choose to use the object creation wizard, proceed as follows:

1. Click Create a new object on the Objects page. Enter the object's name, description and the entity type (for example, a pump).
2. Choose the data source, which corresponds to the type of object:
   - No external data source — For a generic object.
   - Some other system contacts my service to feed the data in (server) — For a server API object.
   - My system will listen to a data channel and topic for incoming data (subscriber) — For a hardware object.
   - My system contacts some other system for data (client) — For a network client object.
   - The data will be entered to a database for my system to read (database)— For a database object.
3. Fill in the parameters for the chosen type of object (see the sections below).
4. Define the access rights (see 4.5, “Managing object access rights”) and click Finish.
5. Add a schema that describes how to parse the data. Schema is created for a particular object type, so choose the type of object you just created in the menu on the left and go to its schemas. For example, Server API objects > API schemas.

> **Note**
>
> This document doesn't provide information on how to write a schema. Please refer to the Asema IoT Central Object data parsing and serializing manual for details.

The object you created appears in the object list Control and Monitor menu. If you want to access the object settings, choose the object type in the Objects menu (see 4.3.2, “Using object forms”).

You can proceed to configuring object properties (see 4.4, “Configuring an object”).

4.3.2. Using object forms

Object forms are used for creating objects or editing the object settings. The forms are type-specific: Generic objects, Server API objects, Network client objects, Database objects, Hardware objects. They can be accessed from the Objects menu.

The data of the object is split into read-only metadata and read-write properties. The fields of the metadata originate from various sources, including those mandated by a driver or object type. In addition, the object may have a metadata model which is a set of metadata fields for the object. If you need certain set of metadata about the objects, you can set up a metadata model in the Objects > Metadata models.

Each object also has a schema that defines what properties the object carries as well as much of the functionality. When you create an object with a form, you must link the object to a schema for the object to become valid (in the case of network client objects you actually need two schemas).

The content and syntax of schemas is a topic covered by the separate Asema IoT Central Object data parsing and serializing manual. This manual does not therefore describe the schemas in further detail.

In all object forms you may add the following identifying details (see above for their meaning):

- Name
- Description
- Smart API object Identifier
- Component Identifier
- Metadata model

Additionally, you can write the object description. This is just a free-form text field that can contain any text you may find useful in remembering or communicating what this object is about.

The rest of the available fields depend on the object type.

4.3.2.1. Generic objects

Because the generic objects do not have an interface, they have a minimal number of object specific settings. The only custom setting is the schema chosen from the corresponding dropdown.

4.3.2.2. Server API objects

To set up a Server API object, you need to define:

- Protocol. This is the network protocol used by an external client to feed data to this object. The following choices are available: HTTP(S), CaAP(S), MQTT and TCP.
- Path identifier, topic or port. Depending on the protocol you've chosen, you need to specify some way to identify which access belongs to which Server API object
  - For HTTP and CoAP, it is a URL path. So if you enter here the value "path_to_my_object", this object (assuming localhost IP 127.0.0.1) can be addressed with http://127.0.0.1:8080/feed/path_to_my_object. Note the mandatory `feed`path needed by the system to recognize access to the Feed API that serves these objects.

- For MQTT, it is the topic. So if you enter here "iot/data/my_object", Asema IoT contacts the default MQTT broker and subscribes to the topic "iot/data/my_object". Messages received from that topic are sent to this object.
- For TCP, it is the port number. So if you enter here 5555, Asema IoT will open this port for listening and data to TCP connection 127.0.0.1:5555 will be sent to this object.

- Core type. This is the most basic classification of an object (you can make a much more detailed one with metadata). Simply says whether the object just reads values (a sensor) or can also control something (an actuator).
- Schema. The schema used for defining properties as well as their parsing from the incoming stream of data.

4.3.2.3. Network client objects

Network client objects are used when Asema IoT acts as a client that actively fetches the data. Typically in this case you are dealing with a network server with a public API that responds to a data query.

To set up a Network client object, you need define:

- Protocol. This is the network protocol used by the Asema IoT system to fetch to this object. The following choices are available: HTTP(S), CaAP, and TCP.
- Target server address. This is the IP or domain address of the server to contact.
- A path identifier or port. Depending on the protocol you've chosen, you need to specify some way to identify how in more detail to address the target server
  - For HTTP and CoAP, it is a URL path. So if you enter here the value "/json/my_object", this object (assuming server data.myiot.org) will be addressed with http://data.myiot.org/json/my_object.
  - For TCP, it is the port number. So if you enter here 5555, Asema IoT will try to contact a TCP port at data.myiot.org:5555.
- Core type. This is the most basic classification of an object (you can make a much more detailed one with metadata). Simply says whether the object just reads values (a sensor) or can also control something (an actuator).
- Active. Unticking this box disables the fetching timer and the object no longer tries to update its values.
- Querying of values. This setting determines what triggers the fetching of values from the remote server
  - On request. The query is made whenever someone tries to read a value of a property of this object.
  - On timer. The values are queried regularly with a timer.
- Schemas. As a network call involves two messages, a request and a response, a network client uses two schemas, one for each.

> **Note**
>
> If you are using a schema that defines Smart API as the standard data format, you may also include a value in the Smart API server identifier, which will identify the Smart API action used at the target server.

4.3.2.4. Database objects

To set up a Database object, you need to define:

- Database identifier. This is the value in a database row or column that identifies the data of this particular object (it will be the value of a WHERE clause). The column itself is defined in the schema.
- Core type. This is the most basic classification of an object (you can make a much more detailed one with metadata). Simply says whether the object just reads values (a sensor) or can also control something (an actuator).
- Querying of values. This setting determines what triggers the fecthing of values from the database:
  - On request. The query is made whenever someone tries to read a value of a proprty of this object.
  - On timer. The values are queried regularly with a timer.
  - On trigger. The fetching is made when the database sends a trigger signal that data in the database has changed.
- Schema. The schema used for defining properties as well as their parsing from the database table.

> **Important**
>
> Triggers work only if you use Postgreql as your database.

4.3.3. Scanning and adding hardware

Hardware objects are directly connected to the computer or device where Asema IoT is running. The setup process depends on the technology used for the connection. For wired protocols you need to type in the details by hand, whereas the wireless protocols support scanning.

4.3.3.1. Bluetooth

There are two major flavors of Bluetooth available: Bluetooth Classic and Bluetooth Smart (aka Bluetooth 4.0 and Bluetooth LE). Because Bluetooth devices emit a beacon signal, adding a Bluetooth device into Asema IoT is a matter of simple scanning. However, you must choose the correct Bluetooth standard to perform the scan in. Bluetooth Smart devices can operate either as connected devices or as simple data transmitters where the data is included in the beacon itself. These beacon only devices are usually so called Bluetooth tags. There are two major standards for the data in the tag beacon: Eddystone and iBeacon. If you are using tags, please also choose the correct beacon format when initiating the scan.

> **Note**
>
> There are several ways a Bluetooth device can communicate with its counterparty. These are defined as Bluetooth profiles. For Bluetooth Classic, Asema IoT always uses the RF Comm profile, making it a device that sends and receives serial data. The schema for Bluetooth then defines how this serial data is interpreted.

To be able to use hardware over Bluetooth, you need to have a computer with Bluetooth radio and must tell the system which Bluetooth adapter you want to use. To do so, enter your Bluetooth adapter MAC address in Settings > Bluetooth. Note that if you want to use a particular Bluetooth standard (Classic or Smart), your Bluetooth hardware must be compatible with that particular standard. Most modern Bluetooth chips support both but not all of them.

To do the scan, go to Objects > Bluetooth and click Add new. Choose the appropriate scan and start it. Once a beacon from your hardware device is detected, the device is shown in the scan results. Click Save to finish the process.

Important

On Linux systems the beacon scanning is done by a background daemon called `proximity_listener`. To be able to do the scan, please ensure this daemon is running.

4.3.3.2. NFC

NFC (Near Field Communication) is a very short range radio standard found today commonly in payment cards, travel cards key fobs, and smart phones. While NFC itself is a standard radio format, there are several substandards to it, each suitable for a particular application such as credit card payments. Asema IoT does not support these substandards but does recognize all NFC cards and can extract the basic information of the cards such as the card number.

To use NFC, you need to have a device with NFC capability. On smartphones and tablets an NFC chip and antenna is typically embedded in the casing of the device. On PC's you will typically need a USB connected card reader.

To connect a device with NFC, go to Hardware objects > NFC and click Add new. Activate the cardreader by clicking Start scan. The scanned device becomes an object in Asema IoT system.

Important

On Linux systems the NFC scanning is done by a background daemon called `proximity_listener`. To be able to do the scan, please ensure this daemon is running.

4.3.3.3. Modbus

Modbus is a serial standard commonly found in industrial devices. Modbus can be used either with a direct serial connection (RTU) or over an IP network (TCP). In either case, you need to manually configure the connection parameters and write a schema that fits the bit/byte order of the Modbus-compatible device you are using.

If you use TCP, type in the IP or domain address of the Modbus device, together with the port.

If you use RTU, enter the serial communication parameters (Baud rate, Data bits, Parity, Stop bits, and Flow control) that match those used by the serial connection of your Modbus device. Also, enter the correct serial port to use into the Address field.

Note

For details on the Modbus schema and how to write one, refer to the Asema IoT Central Object data parsing and serializing manual. As an alternative to a schema, you can write a completely custom parser plugin that does the conversion.

4.3.3.4. Serial

Serial devices simply transmit raw bytes over a wire and interpret the bytes as property values. The interpretation is done by a parsed with the help of a schema that customizes the parser. To use a serial

device, connect it to a serial port and type the name of the serial port into the Address field of the configuration form. Then enter the serial communication parameters (Baud rate, Data bits, Parity, Stop bits, and Flow control) and make sure they match those used your serial device

Note

For details on the serial schema and how to write one, read the Asema IoT Central Parser and serializer manual. As an alternative to a schema, you can also write a completely custom parser plugin that does the conversion.

4.3.3.5. Custom hardware drivers

If you have a hardware device that does not conform to any of the models readily available in Asema IoT, you can program a driver for it as a plugin that is completely custom. The plugin is a C/C++ program compiled as a dynamically linkable library (a .dll or .so) which you Asema IoT loads at startup and connects into the object framework.

This manual doesn't provide information on how to write those plugins. For details on plugin programming, including custom hardware driver plugins, refer to the Asema IoT Central Plugin programming manual.

Once you programmed a plugin, add it to the system by placing it into the plugins directory of your installation (see the Asema IoT Central Plugin programming manual for details). Then restart the system, go to Hardware objects > Custom and click Add new. If the plugin is programmed correctly, you can choose the driver from the dropdown menu here.

4.4. Configuring an object

Once you created an object, you can further configure its behavior by editing object metadata, configuring how it writes measurement values into the database, and whether and how notifications of changes into properties are sent forward.

4.4.1. Object metadata

If there is any object metadata that should be used by the Asema IoT Central or by the device driver, specify it on the Objects > Configure page. Click the gear icon in the Metadata column and fill in the fields:

- Common metadata — Includes the the object location.
- Custom metadata — Any kind of information about the object that you need to specify. For example, the device model. To enter it, add a text field "Device model" and then enter the model name as a value.
- Extended metadata — Information used by the hardware object's driver (if you programmed one).
- Default property values — Values used by the plugin (if you programmed one).

4.4.1.1. Setting up the metadata model

If you have multiple objects of the same kind (let's say, cars for rent), you might need a custom form to to store the necessary information about each of them. To do it, create a custom form with the necessary fields (such as, for example, the manufacturer, production year, number of seats and so on), link it to the object and enter the details in it.

1. Go to Objects > Metadata models and click Add new.

2. Enter the name for the metadata model.

3. Create the metadata fields: choose the field type and label.

4. Save the data model.

Now you can link the form to the objects you create.

Once you entered the meta data when creating the object, you can edit in in Objects > Configure > Metadata > Custom metadata.

4.4.2. Database logging

Asema IoT can record the values received from the object to the database.

To set up database logging for an object, go to Objects > Configure and click the gear icon in the Database logging column. Choose the logging options and click Save.

4.4.3. Notifications sending

To send the object's data to the MQTT broker server, specify the server and the topic in the object's settings.

To do this, go to Objects > Configure and click the gear icon in the Notifier column. Choose a notification broker in the Server field (the server must be present in System > Notification sender). Specify the message format and the three parts of the topic name (Publisher type, Conversation ID, Conversation type).

4.4.4. Subscriber settings

If you are using Asema E gateways, those gateways will send notifications of property values only when the values of the properties change. In case no change is detected, the Asema IoT will not receive any values until change. To get regular updates of property values even when values remain static, you need to inform the gateway to do so.

To subscribe to the property on a regular basis, add it to the subscriber settings. The setting is simple, it is just a list of property names, each with its own entry. These properties will get regular updates, others only change notifications.

> **Note**
>
> Subscriber settings are only available for objects that originate from Asema E compatible gateways. On other objects they have no effect.

4.5. Managing object access rights

Depending on the interface the user is working with, you can control the user's access to the objects with the following tools:

- Zone- and user group-based access in graphical interface.
- API key for JSON API.
- Offers in Smart API.

- Sharing objects with peers via Smart API.

4.5.1. Zone- and user group-based access

In graphical interface, the access to object is based on zones and user groups. An object is assigned a zone that limits the access to the specified geographical area and (optionally) the time period. Then, you can set which usergroups have access to the zone.

Thus, object O linked to zone Z can be accessed by the user groups that have access rights for the zone Z.

To configure the object access:

1. Create a zone and configure the zone access rights.

   Go to Objects > Zones and click Add new. Fill in the fields:

   - Name of the zone.
   - Access rights for each user group:

     View — View the object's properties and settings.

     Edit — Edit the object's properties and settings.

     Delete — Delete an object.

     Control — Set the object's property values.

     Record — Record the object's property values.

     Set target — Set target for an object's property.

     Bid — Create offers for buying the object's properties.

     Ask — Create offers for selling the object's properties.
   - Limit to area — geographical area. To add a map area from the Content menu, choose it in the Copy from area list.
   - Valid from, Valid until — The period in which the specified user groups can access the zone's objects.
   - Description of the zone.

2. Assign the zone to the object.

   Go to Objects > Configure and click the gear button in the Access control field next to the object. Tick the Enable zone based access control for this object option, choose the zone and click Save.

3. Turn on zone-based access to the object.

   Go to Objects > Configure and put the Access control switcher next to the object to ON.

4.5.2. Access via JSON API

You can define if the object is available via the JSON API.

To do this, go to Objects > Configure and click the gear button in the Access control field next to the object. Set the access level in the External access with generic API keys section and enter the API keys.

4.5.3. Access via Smart API

Via Smart API, access to objects can be bought when the user receives an offer. To enable the offering protocol for an object, go to Objects > Configure, click the gear button in the Access control field next to the object and choose Offering protocol enabled in the External access through Smart API section.

4.5.4. Sharing settings

To publish an object for sharing to other systems, you need to open peer access to it. Choose one of the options:

- Sharing disabled. Not visible in Smart API Find searches.
- Sharing enabled. Always visible in Smart API Find searches.
- Sharing enabled. Visible in Smart API Find searches only when an offer is in force.

For more information about sharing functionality and settings, see the Sharing chapter in Asema IoT Central user manual.

4.6. Using the control dashboard and graphing values

To track the object data and control the object, go to Objects > Control and Monitor.

Here you can:

- Get information about the object.

  The information about the objects is displayed in the table on the Objects page. You can see the object source (if the object is local, pooled or shared) and connection status (online or offline).

  To view the object identifiers [9], click the information icon next to the object.

- Check if the object is online.

  To do this, click Ping. If the object is connected, it shows a heartbeat sign. If it is disconnected, an error is displayed.

- Create events and tasks related to the object. To do this, click the gear icon in the Events or Tasks column and fill in the fields. Events and tasks are displayed on the Object events timeline.

- Generate reports in XLS and PDF format (see the Generating reports chapter in the Asema IoT Central user manual).

- Set up a dashboard for monitoring and controlling the object.

  To create a dashboard, click the gauge icon next to the object to go to the dashboard page. Then click the button in the upper right corner of the page to add elements and controls, such as gauges, charts, property setters and sliders, maps and so on.

  To save the dashboard, click the save icon at the top of the page.

4.7. Importing and exporting objects

You can import and export objects between different Asema IoT instances. Object information is saved in the JSON format and can be downloaded in a text file. To export or import an object, go to Objects > Import/Export.

4.8. Deploying objects with Asema IoT Central mobile app

Asema IoT allows you to add remote objects to your system using the Asema IoT mobile app. It is especially convenient if you need to connect multiple remote devices located in the field. Imagine you have a company maintaining windmills and you need to monitor hundreds of windmills in various locations. You want to add them to the system, and for each windmill, you want to have coordinates. manufacturer and the product ID. All of these can be obtained only on the spot. To do it, you would need to do the following:

1. Feed the list of objects in Asema IoT. You can do it through the API or on the Objects > Import, export, deploy > Import placeholders from file page.
2. Add the object schema and specification in JSON on the Objects > Import, export, deploy > Manage undeployed page.
3. Create a metadata model and link it to the deployment form on the Objects > Import, export, deploy > Manage undeployed page.
4. Add the deployment instruction on the Objects > Import, export, deploy > Manage undeployed page. This instruction will be shown in Asema IoT Dashboard app.
5. Print out the QR codes for the objects (optionally, if you want to avoid searching for the object in the list).
6. On the spot where the object is located, stick the QR code to the object and then scan it with Asema IoT Dashboard app. The app will show the instruction and prompt you to fill in the meta data fields. After deployment is completed, the object will appear in Asema IoT.

4.9. Managing connected hardware

The list of connected hardware is available on the Objects > Hardware objects > Manage page.

To delete a connected device, click the trash icon.

To reload all drivers, click Reconnect.

5. Rule automation

The rule engine lets you automate the system's operation. The basic concept of a rule is straightforward: each rule says that **if** an event takes place, **then** perform an action. Once started, rules "run" inside the rule engine i.e. are evaluated automatically without further interaction. All you have to do is to specify what has to be done in particular situations. For example, "If it is sunset, turn on the lights" or "If the temperature is below 18 degrees, turn on the heating".

As you can see from the examples, a rule consists of two parts, the condition ("if") and the consequence ("then"). Building rules means designing both conditions and consequences and then linking them together. When conditions and consequences are linked, conditions send signals to consequences. A condition that becomes true (or false if it is a negated condition) is said to trigger. This triggering causes either a true or false signal to be sent. The consequence is then set up to react to that particular signal.

Depending on the complexity of the rule you need to compose, you can use one of the tools:

- Standard rules with boolean logic. These rules are built from ready-made blocks in the Asema IoT web admin interface.
- Custom rules. These rules can have more complex logic and are programmed with JavaScript.

5.1. Standard rules

A standard rule in the Asema IoT engine has two basic components:

- Conditions, i.e. the condition that triggers the rule. There are several types of conditions
  - Timer conditions — set the time to perform an action.
  - Property conditions — set the threshold value of the object property (for example, "if the unit temperature rises to 60 degrees").
- Consequences, i.e. the action to perform if a condition comes true (for example, "display a warning message").

A rule is then simply a combination of conditions and consequences, for example: "if property A of object O_1 reaches value X then set property B of object O_2 to value Y."

A rule may have multiple conditions and multiple consequences. The logical rule between multiple conditions is always AND. If you need other logical operations (i.e. OR and NOT), these are not directly supported but can be easily programmed with scriptable rules (see below). Multiple conditions are evaluated in sequence, i.e. if the first condition becomes true, the second condition is evaluated, then the third condition, and so on. Once the final condition in the chain becomes true, then the rule triggers.

Multiple consequences in a rule are run at a non-deterministic random order. So if you set two consequences, one that subtracts a value of a property and another that adds a value to the same property, there is no guarantee that the subtraction takes place before the addition. It is good to bear in mind that while the arithmetic end result in this case is the same either way (addition and subtraction are commutative), setting a property may have an impact in some device that is controlled by the object and its property. Setting things in the wrong order may cause, for instance, overflows or malfunctions. If this is an issue, it is recommended to either use just single consequence per rule and chain two rules together or use scripted rules.

Also, a rule can include a negative condition: "if a given value is measured, don't do anything".

5.1.1. Creating a timer condition

Timer condition allows you to set the time to perform an action. You can set the timer to particular time and date or a sunrise or sunset event. A timer condition can be repeated daily or limited to specific days of the week. To create a timer condition:

1. Go to Automation > Timer conditions. Click the "plus" icon and then the "gear" icon.
2. Choose the timer type:
   - Clock — Sets the time for the action and (optionally) repeats it daily or on certain days of week.
   - Clock and calendar — Sets the time and date for the action, can't be repeated.
   - Sunrise/ sunset — Set the time to sunrise, sunset or noon based on the system location. Can be repeated daily.
3. Fill in the fields for the chosen timer type.

   For all timer types, you can set a deviation, that means make the action happen within the specified time frame, for example, 600 seconds before or after sunset. This option is sometimes helpful if you don't want the rule to be triggered always at the same time.
4. Save the timer condition.

To complete the automation, you need to add a consequence and define a rule.

> **Important**
>
> Sunrise and sunset calculation rely in the position information of your system. Define in in System settings for the correct calculation of this timer.

5.1.2. Creating a property condition

Property condition lets you create rules that are triggered when an object property takes a certain value. For example, if you need to turn on the heating when the room temperature drops below certain point, set the temperature threshold in the condition.

To create a condition:

1. Go to Automation > Property conditions. Click the plus icon and then the gear icon.
2. Fill in the fields:

   Condition name — Can be anything as long as it is unique.

   Target object — The sensor or other unit connected to the source device you want to monitor.

   Property to monitor — The property to track.

   Value limit — The property value which should trigger the consequence.

   Condition type — The type of event that makes the condition valid. For example, "value is greater than <the value threshold>".

   React to changes only — Whether the condition should be valid only when the tracked property value changes (as opposed to triggering the condition every time the value is updated, even if it has the same value as before).
3. Save the property condition.

To complete the automation, you need to add a consequence and define a rule.

5.1.3. Creating a consequence

Consequences are actions that take place once the condition is fulfilled. To define a consequence:

1. Go to Automation > Consequences. Click the plus icon and then the gear icon.

2. Fill in the fields:

   Consequence name — Can be anything as long as it is unique.

   Target object — A connected unit that should react if the condition is valid.

   Action to perform — An action that should happen if the condition is valid:

   - Set controller on/off — Turn the controller type object on or off.
   - Set controller to value — Change the controller_property value of the controller type object.
   - Run an action — Perform an action programmed in Asema E application (you need to use Asema E as a gateway). Actions in Asema E enable you to control multiple units at once.
   - Display message — Display message on the screen.
   - Add, subtract, multiply, divide the property value.
   - Send an email — Send an email to the specified email address. The email template is set up in the System > Email section.

3. Save the consequence.

5.1.4. Setting up and testing rules

To make a rule, put together conditions and consequences you created:

1. Go to Automation > Rules and click Add new standard rule.

2. Fill in the fields:

   - Name — The rule name.
   - Triggering conditions — Conditions that trigger the consequence actions.
   - Blocking conditions — Conditions that block the implementation of the rule even if the triggering condition takes place.
   - Consequence — Action to implement if the triggering condition takes place. Choose the consequence from the list.
   - Reacts when — Connection between the condition and consequence. In most cases, the rule assumes that the consequence should be implemented when the condition is true (the "Condition is true/start" option). However, if you are setting up a rule based on the tariff condition and two consequences (such as, "if the battery is low, turn the charger on, if the battery is full, turn the charger off"), add the consequences and specify when each one of them is used ("Condition is true/start" and "Condition is false/stop").

To test how the rule works, go to Automation > Rule tester. Change the condition values. When the consequence is triggered, you will see the message "Triggered rule <rule name>" at the top right of the screen.

5.2. Programming rules with JavaScript

5.2.1. Scriptable rules

Scriptable rules are rules written on JavaScript to perform custom analytics of events that take place in the Asema IoT system. The JavaScript is wrapped inside tags that make it valid QML and therefore

capable of receiving and sending events to other objects through the internal JavaScript engine of an Asema IoT system. As the wrapper of the script is QML, you can also add QML elements useful for such programming such as Timers.

A scripted rule can perform both the condition part and the consequence in the JavaScript itself. This way the rule is a self-contained routine inside the rule engine and does not interact with anything else. Alternatively, the JavaScript section is just the conditional part of the rule. It is quite common that the evaluation of rule conditions requires more logic while the actual consequence is simple. For instance, turning on an AC may require a careful analysis of several variables such as temperature, humidity and time of day but the actual action to take is just turning on the device. This is why the script can just emit a signal then it triggers and the action can be linked to it as a standard consequence.

A JavaScript rule must begin with the import statement `import RuleEngine 1.0` that imports the necessary hooks to the JavaScript engine. The rest of the rule must then be wrapped inside a `RuleScript` element. Please see sample code below.

Inside the rule there are two mandatory elements:

- The signal `result`.
- The function `init()`.

`result` is the signal emitted once the rule triggers, i.e. wants to signal the engine that now the system should react to something. `init()` is a method guaranteed to be called by the rule engine once the rule has completed loading into the engine. It should contain all the initialization routines and work as the startup routine that actually runs the logic you have programmed.

Note that the rule engine does not call any other functions than `init()` at the very beginning when the rule is loaded into memory. Specifically: there is no method called by the engine when it wants the rule to be evaluated. Instead, the rest of the rule logic must work on events that you receive from objects. You need to program the rule so that the `init()` loads the objects you are interested in and then the rule connects to events such as `measurableValueChanged` of the object. This way your rule will react to changes in objects and their states immediately when events take place. Instead of relying on some method to be called at some interval, the rule will now be able to react in real-time to **any** change in objects that may take place.

To load objects during init, you need to find them. This is done using `find` methods of the `objectManager` (which is a part of the rule engine), including `findObjectsByType` and `findObjectsByName`. These operations are asynchronous. Before you call a find method, you must therefore define a callback which is called once the load completes. Once loaded, the objects are stored into a property of the context of your rule from where the objects can be fetched. The default context `context` is always available for this purpose. For instance, the call `objectManager.findObjectsByType(context, "myAppliances", CoreObject.ApplianceObject);` will use the `objectManager` to find all objects of the `CoreObject.Appliance` type and then store them to a property of `context` called `myAppliances`.

> **Note**
>
> The `objectManager` is exactly the same component that is used when programming custom applications, such as screenlets. For more details on how to use the `objectManager`, including programming syntax and the API, please refer to the application programming section of the Asema IoT Central user manual.

Once you have a list of objects from `objectManager`, they can be connected to the rule logic. For example, `context.objects.myAppliances.list[0].stateChanged.connect(applianceStateChanged)` will take the first object in `myAppliances` and connect the `stateChanged` event into the function called `applianceStateChanged`.

`applianceStateChanged` should then contain the actual rule logic, i.e. what should be done when the state of that particular appliance changes. The event could, for instance, start a timer or a measurement or test what the state is now and then simply signal that the rule just triggered.

5.2.2. Writing a rule script

Below is an example rule which loads a set of appliances and sensors into the rule. Appliance state changes cause no real effect, they just log messages into the system log. Changes in sensor values are on the other hand compared to a threshold and if that is exceeded, the rule triggers.

```
import RuleEngine 1.0

RuleScript {
    id: myTestRule

    signal result

    function init() {
        console.log("MyTestRule is initing");

        myTimer.running = true;

        console.log("Example: Load the objects and connect to an object")
        context.objectsReady.connect(connectToAppliance);

        console.log("Example: Find all appliances")
        objectManager.findObjectsByType(context, "myAppliances", CoreObject.ApplianceObject);

        console.log("Example: Find a particular appliance by name")
        objectManager.findObjectsByName(context, "myAirQuality", "White air");

        return "init OK";
    }

    function timerTick() {
        console.log("Tic! Toc! The timer just ticked.");
    }

    function connectToAppliance(property) {
        console.log("Property \"" + property + "\" is ready.");

        if (property == "myAppliances") {
            console.log("A total of " + context.objects.myAppliances.list.length + " items found");
            console.log("The first object is called " + context.objects.myAppliances.list[0].name);
            console.log("Example: Connect to a state change of appliance.");
            context.objects.myAppliances.list[0].stateChanged.connect(applianceStateChanged);
        } else if (property == "myAirQuality") {
            console.log(context.objects.myAirQuality.list.length + " items found with given name.");
            context.objects.myAirQuality.list[0].measurableValueChanged.connect(airQualityChanged);
        }
    }

    function applianceStateChanged(state) {
        console.log("Appliance state changed to " + state);
    }

    function airQualityChanged(name, value) {
        console.log("Sensor value " + name + " changed to " + value);
        if (name == "temperature_celsius" && value*2 > 80) {
            console.log("Trigger the rule");
            myTestRule.trigger();
            console.log("Done");
        }
    }

    Timer {
        id: myTimer
        interval: 10000

        onTriggered: {
            timerTick();
        }
    }
}
```

6. Users and access rights

In Asema IoT, access rights are defined with user groups and object zones. A user group is,as the name implies, a group of users. An object zone on the other hand is a group of objects. Theintersection of these two gives rights to operate an object.

Take an example: we have campus of several buildings. Each building has several remotely controlled relays. They are managed by two building managers that are assigned to a particular building. To give access to the relays of a particular building to a particular person, access rights would be set as follows: each building is a zone, so the relays of building A would belong to Zone A. The personnel belongs to groups, say Building A Managers, Building B Managers, and so on. Now, to grant access to the relays in building A to its managers, the access control would say "grant rights to objects of Zone A to the user group Building A Managers".

6.1. Add users

To add a user, go to System > Users > Users menu and click Add new. Fill in the fields, assign a groupto the user, then click Save.

> **Note**
>
> The user won't be able to change the password you provided unless they belong to the group that has access to user management menu.

6.2. Manage groups

By default, Asema IoT features two user groups, "everyone" and "admin". You can add as many additional groups as needed. Note that the scope of features associated with the admin group cannot be customized. This is to ensure that admins do not accidentally lose their ability to configure the system. The system requires that there always is at least one admin user.

You can create new groups and edit the list of features available to each group.

To create a group, click Add new, enter the group name, choose the features that should be availabe to the group and click Save.

To edit the list of features a group has access to, go to the System > Users > Groups menu, choose agroup and tick or untick the features, such as "Edit objects", "Edit content" and so on.

6.3. Restore the admin user password

If the admin password of the system was lost, it can be reset. To reset the admin password, you need to have access to the server where Asema IoT Central is running. To restore access:

1. Quit/ stop Asema IoT.
2. Start the program with the -P flag and with the new password as the argument. E.g. open the command line and enter the command:

   ```
   asema_iot_central.sh -P myNewAdminPassword
   ```

> **Note**
>
> It is not possible to recover a lost admin password from Asema IoT web interface. To change a lost admin password, you need to have admin/supervisor access to the actual computer that runs Asema IoT and use command line there.

Asema Electronics Ltd