Eclipse Kura
MQTT Namespace Guidelines

Overview
MQTT Request/Response Conversations
- MQTT Request/Response Example

MQTT Remote Resource Management
- Read Resources
- Create or Update Resources
- Delete Resources
- Other Operations

MQTT Unsolicited Events
Discoverability
Remote OSGi Management via MQTT
- Remote OSGi ConfigurationAdmin Interactions via MQTT
  - Read All Configurations
  - Read Configuration for a Given Service
  - Update All Configurations
  - Update the Configuration of a Given Service
  - Example Management Web Application
- Remote OSGi DeploymentAdmin Interactions via MQTT
  - Read All Deployment Packages
  - Install a Deployment Package
  - Uninstall a Deployment Package
  - Read All Bundles
  - Start a Bundle
  - Stop a Bundle
  - Example Management Web Application

Overview

This section provides guidelines on how to structure the MQTT topic namespace for messaging interactions with applications running on IoT gateway devices.

Interactions may be solicited by a remote server to the gateway using a request/response messaging model, or unsolicited when the gateway simply reports messages or events to a remote server based on periodic or event-driven patterns.

The table below defines some basic terms used in this document:

account_name	Identifies a group of devices and users. It can be seen as partition of the MQTT topic namespace.

	For example, access control lists can be defined so that users are only given access to the child topics of a given account_name.
client_id	Identifies a single gateway device within an account (typically the MAC address of a gateway's primary network interface). The **client_id** maps to the Client Identifier (Client ID) as defined in the MQTT specifications.
app_id	Identifies an application running on the gateway device. To support multiple versions of the application, it is recommended that a version number be assigned with the **app_id** (e.g., "CONF-V1", "CONF-V2", etc.)
resource_id	Identifies a resource(s) that is owned and managed by a particular application. Management of resources (e.g., sensors, actuators, local files, or configuration options) includes listing them, reading the latest value, or updating them to a new value. A **resource_id** may be a hierarchical topic, where, for example, "sensors/temp" may identify a temperature sensor and "sensor/hum" a humidity sensor.

A gateway, as identified by a specific **client_id** and belonging to a particular **account_name,** may have one or more applications running on it (e.g., "app_id1", "app_id2", etc.). Each application can manage one or more resources identified by a distinct **resource_id**(s).

Based on this criterion, an IoT application running on an IoT gateway may be viewed in terms of the resources it owns and manages as well as the unsolicited events it reports.

MQTT Request/Response Conversations

Solicited interactions require a request/response message pattern to be established over MQTT. To initiate a solicited conversation, a remote server first sends a request message to a given application running on a specific device and then waits for a response.

To ensure the delivery of request messages, applications that support request/response conversations via MQTT should subscribe to the following topic on startup:

$EDC/account_name/client_id/app_id/#

The **$EDC** prefix is used to mark topics that are used as *control topics* for remote management. This prefix distinguishes control topics from data topics that are used in unsolicited reports and marks the associated messages as transient (not to be stored in the historical data archive, if present).

NOTE: While Eclipse Kura currently requires "$EDC" as the prefix for control topics, this prefix may change in the future for the following reasons:

- MQTT 3.1.1 discourages the use of topic starting with "$" for application purposes.
- As a binding of LWM2M over MQTT is taking shape, it would makes sense to use a topic prefix for management messages like "LWM2M" or similar abbreviations – e.g. "LW2", "LWM".

A requester (i.e., the remote server) initiates a request/response conversation through the following events:

1. Generating a conversation identifier known as a **request.id** (e.g., by concatenating a random number to a timestamp)

2. Subscribing to the topic where the response message will be published, where **requester.client.id** is the client ID of the requester, such as:

   $EDC/account_name/requester.client.id/app_id/REPLY/request.id

3. Sending the request message to the appropriate application-specific topic with the following fields in the payload:

   - **request.id** (identifier used to match a response with a request)
   - **requester.client.id** (client ID of the requester)

The application receives the request, processes it, and responds on a REPLY topic structured as:

$EDC/account_name/requester.client.id/app_id/REPLY/request.id

NOTE: While this recommendation does not mandate the format of the message payload, which is application-specific, it is important that the **request.id** and **requester.client.id** fields are included in the payload. Eclipse Kura leverages an MQTT payload encoded through Google Protocol Buffers. Eclipse Kura includes the request.id and the requester.client.id as two named metrics of the Request messages. The Eclipse Kura payload definition can be found at the following link:

- https://github.com/eclipse/kura/blob/develop/kura/org.eclipse.kura.core.cloud/src/main/protobuf/kurapayload.proto

Once the response for a given request is received, the requester unsubscribes from the REPLY topic.

MQTT Request/Response Example

The following sample request/response conversation shows the device configuration being provided for an application:

account_name: guest
device client_id: F0:DE:F1:C4:53:DB
app_id: CONF-V1
Remote Service Requester client_id: 00:E0:C7:01:02:03

The remote server publishes a request message similar to the following:

- Request Topic:
  - **$EDC/guest/F0:DE:F1:C4:53:DB/CONF-V1/GET/configurations**

- Request Payload:
  - **request.id: 1363603920892-8078887174204257595**
  - **requester.client.id: 00:E0:C7:01:02:03**

The gateway device replies with a response message similar to the following:

- Response Topic:
  - **$EDC/guest/00:E0:C7:01:02:03/CONF-V1/REPLY/1363603920892-8078887174204257595**

- Response Payload, where the following properties are mandatory:
  - **response.code**

    Possible response code values include:
    - **200 (RESPONSE_CODE_OK)**
    - **400 (RESPONSE_CODE_BAD_REQUEST)**
    - **404 (RESPONSE_CODE_NOTFOUND)**
    - **500 (RESPONSE_CODE_ERROR)**

  - **response.exception.message** (value is null or an exception message)
  - **response.exception.message** (value is null or an exception stack trace)

NOTE: In addition to the mandatory properties, the response payload may also have custom properties whose description is beyond the scope of this document.

It is recommended that the requester server employs a timeout to control the length of time that it waits for a response from the gateway device. If a response is not received within the timeout interval, the server can expect that either the device or the application is offline.

MQTT Remote Resource Management

A remote server interacts with the application's resources through *read*, *create* and *update*, *delete,* and *execute* operations. These operations are based on the previously described request/response conversations.

Read Resources

An MQTT message published on the following topic is a read request for the resource identified by the **resource_id**:

- **$EDC/account_name/client_id/app_id/GET/resource_id**

The receiving application responds with a REPLY message containing the latest value of the requested resource.

The **resource_id** is application specific and may be a hierarchical topic. It is recommended to design resource identifiers following the best practices established for REST API.

For example, if an application is managing a set of sensors, a read request issued to the topic "**$EDC/account_name/client_id/app_id/GET/sensors**" will reply with the latest values for all sensors.

Similarly, a read request issued to the topic "**$EDC/account_name/client_id/app_id/GET/sensors/temp**" will reply with the latest value for only a temperature sensor that is being managed by the application.

Create or Update Resources

An MQTT message published on the following topic is a create or update request for the resource identified by the **resource_id**:

- **$EDC/account_name/client_id/app_id/PUT/resource_id**

The receiving application creates the specified resource (or updates it if it already exists) with the value supplied in the message payload and responds with a REPLY message.

As in the read operations, the **resource_id** is application specific and may be a hierarchical topic. It is recommended to design resource identifiers following the best practices established for REST API. For example, to set the value for an actuator, a message can be published to the topic

"**$EDC/account_name/client_id/app_id/PUT/actuator/1**" with the new value supllied in the message payload.

Delete Resources

An MQTT message published on the following topic is a delete request for the resource identified by the **resource_id**:

- **$EDC/account_name/client_id/app_id/DEL/resource_id**

The receiving application deletes the specified resource, if it exists, and responds with a REPLY message.

Execute Resources

An MQTT message published on the following topic is an execute request for the resource identified by the **resource_id**:

- **$EDC/account_name/client_id/app_id/EXEC/resource_id**

The receiving application executes the specified resource, if it exists, and responds with a REPLY message. The semantics of the execute operation is application specific.

Other Operations

The IOT application may respond to certain commands, such as taking a snapshot of its configuration or executing an OS-level command. The following topic namespace is recommended for command operations:

- **$EDC/account_name/client_id/app_id/EXEC/command_name**

An MQTT message published with this topic triggers the execution of the associated command. The EXEC message may contain properties in the MQTT payload that can be used to parameterize the command execution.

MQTT Unsolicited Events

IOT applications have the ability to send unsolicited messages to a remote server using events to periodically report data readings from their resources, or to report special events and observed conditions.

NOTE: It is recommended to not use MQTT control topics for unsolicited events, and subsequently, to avoid the $EDC topic prefix.

Event MQTT topics generally follow the pattern shown below to report unsolicited data observations for a given resource:

- **account_name/client_id/app_id/resource_id**

Discoverability

The MQTT namespace guidelines in this document do not address remote discoverability of a given device's applications and its resources. The described interaction pattern can be easily adopted to define an application whose only responsibility is reporting the device profile in terms of installed applications and available resources.

Remote OSGi Management via MQTT

The concepts previously described have been applied to develop a solution that allows for the remote management of certain aspects of an OSGi container through the MQTT protocol, including:

- Remote deployment of application bundles
- Remote start and stop of services
- Remote read and update of service configurations

The following sections describe the MQTT topic namespaces and the application payloads used to achieve the remote management of an OSGi container via MQTT.

NOTE: For the scope of this document, some aspects concerning the encoding and compressing of the payload are not included.

The applicability of the remote management solution, as inspired by the OSGi component model, can be extended beyond OSGi as the contract with the managing server based on MQTT topics and XML payloads.

Remote OSGi ConfigurationAdmin Interactions via MQTT

An application bundle is installed in the gateway to allow for remote management of the configuration properties of the services running in the OSGi container.

For information about the OSGi Configuration Admin Service and the OSGi Meta Type Service, please refer to the OSGi Service Platform Service Compendium 4.3 Specifications.

The **app_id** for the remote configuration service of an MQTT application is "**CONF-V1**". The resources it manages are the configuration properties of the OSGi services. Service configurations are represented in XML format.

The following service configuration XML message is an example of a WatchDog service:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:configuration xmlns:ns2=http://eurotech.com/esf/2.0
                   xmlns=http://www.osgi.org/xmlns/metatype/v1.2.0
                   pid="org.eclipse.kura.watchdog.WatchdogService">
```

```
 <OCD id="org.eclipse.kura.watchdog.WatchdogService"
     name="WatchdogService"
     description="WatchdogService Configuration">
   <Icon resource="WatchdogService"/>
   <AD id="watchdog.timeout"
      name="watchdog.timeout"
      required="true"
      default="10000"
      cardinality="0"
      type="Integer"
      description=""/>
 </OCD>
 <ns2:properties>
   <ns2:property type="Integer" array="false" name="watchdog.timeout">
     <ns2:value>10000</ns2:value>
   </ns2:property>
 </ns2:properties>
</ns2:configuration>
```

The service configuration XML message is comprised of the following parts:

- The **Object Class Definition** (OCD), which describes the service attributes that may be configured. (The syntax of the OCD element is described in the OSGi Service Platform Service Compendium 4.3 Specifications, Section 105.3.)

- The **properties** element, which contains one or more properties with their associated type and values. The type name must match the name provided in the corresponding attribute definition identifier (AD id) contained in the OCD.

The "CONF-V1" application supports the read and update resource operations as described in the following sections.

Read All Configurations

This operation provides all service configurations for which remote administration is supported.

- Request Topic:
  - **$EDC/account_name/client_id/CONF-V1/GET/configurations**

- Request Payload:
  - Nothing application-specific beyond the request ID and requester client ID

- Response Payload:
  - Configurations of all the registered services serialized in XML format

Read Configuration for a Given Service

This operation provides configurations for a specific service that is identified by an OSGi service persistent identifier (**pid)**.

- Request Topic:
  - **$EDC/account_name/client_id/CONF-V1/GET/configurations/pid**

- Request Payload:
  - Nothing application-specific beyond the request ID and requester client ID

- Response Payload:
  - Configurations of the registered service identified by a **pid** serialized in XML format

Update All Configurations

This operation remotely updates the configuration of a set of services.

- Request Topic:
  - **$EDC/account_name/client_id/CONF-V1/PUT/configurations**

- Request Payload:
  - Service configurations serialized in XML format

- Response Payload:
  - Nothing application-specific beyond the response code

Update the Configuration of a Given Service

This operation remotely updates the configuration of the service identified by a **pid**.

- Request Topic:
  - **$EDC/account_name/client_id/CONF-V1/PUT/configurations/pid**

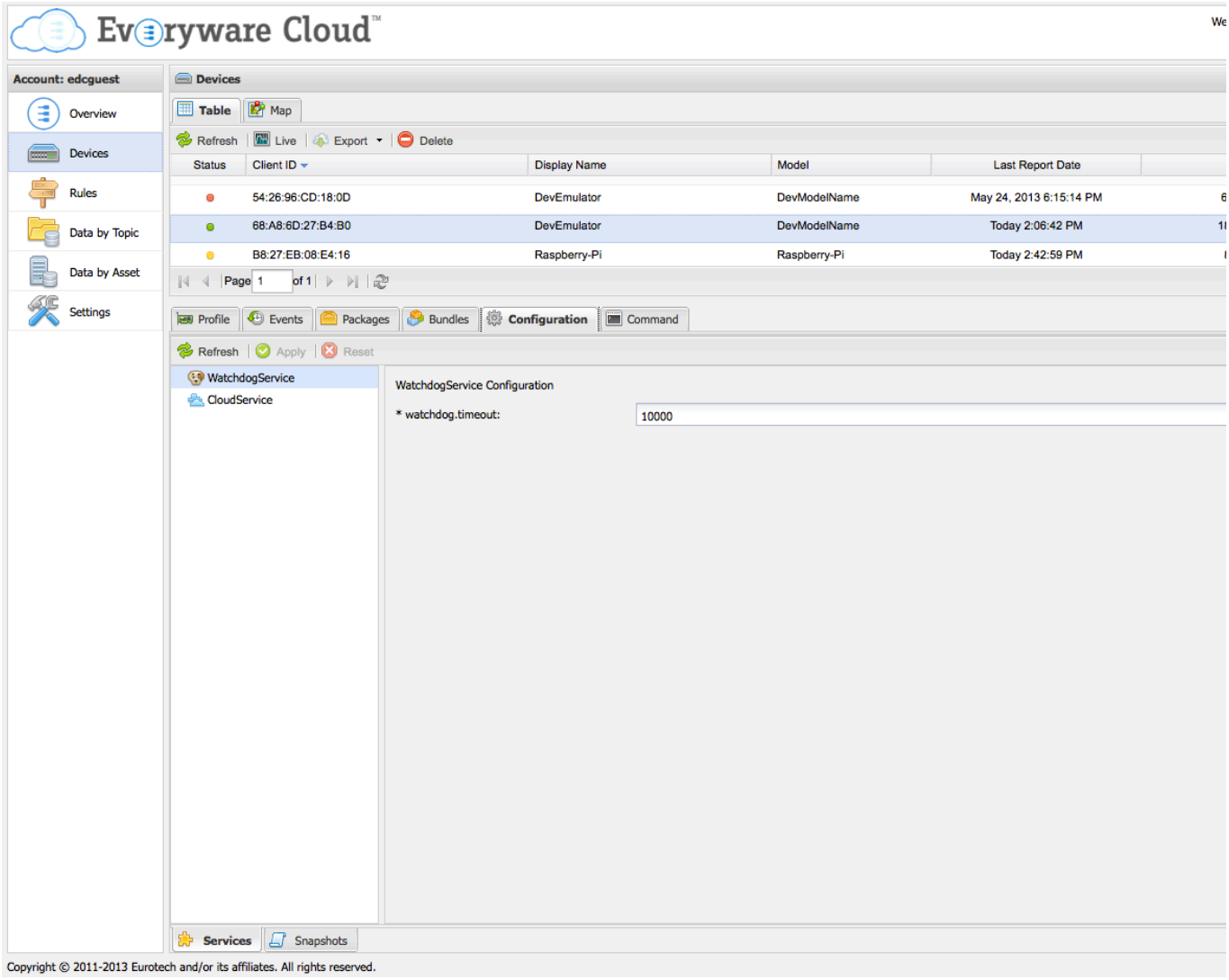
- Request Payload:
  - Service configurations serialized in XML format

- Response Payload:
  - Nothing application-specific

Example Management Web Application

The previously described read and update resource operations can be leveraged to develop a web application that allows for remote OSGi service configuration updates via MQTT though a web user-interface.

The screen capture that follows shows an example administration application where, for a given IOT gateway, a list of all configurable services is presented to the administrator.



When one such service is selected, a form is dynamically generated based on the metadata provided in the service OCD. This form includes logic to handle different attribute types, validate acceptable value ranges, and render optional values as drop-downs. When the form is submitted, the new values are communicated to the device through an MQTT resource update message.

Remote OSGi DeploymentAdmin Interactions via MQTT

An application is installed in the gateway to allow for the remote management of the deployment packages installed in the OSGi container.

For information about the OSGi Deployment Admin Service, please refer to the OSGi Service Platform Service Compendium 4.3 Specifications.

The **app_id** for the remote deployment service of an MQTT application is "**DEPLOY-V1**". The resources it manages are the packages deployed in the OSGi container. Deployment packages are represented in XML format.

The following XML message is an example of a service deployment:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<packages>
   <package>
      <name>esf</name>
      <version>1.0.0</version>
      <bundles>
         <bundle>
            <name>org.hsqldb.hsqldb</name>
            <version>2.2.9</version>
         </bundle>
         <bundle>
            <name>org.eclipse.kura.linux</name>
            <version>1.0.0.SNAPSHOT</version>
         </bundle>
         <bundle>
            <name>javax.bluetooth</name>
            <version>2.1.1</version>
         </bundle>
         <bundle>
            <name>org.eclipse.kura.protocol.modbus</name>
            <version>1.0.0.SNAPSHOT</version>
         </bundle>
         <bundle>
            <name>org.apache.commons.net</name>
            <version>3.1.0.v201205071737</version>
         </bundle>
         <bundle>
            <name>javax.usb.api</name>
            <version>1.0.2</version>
         </bundle>
         <bundle>
            <name>org.apache.servicemix.bundles.protobuf-java</name>
            <version>2.4.1.1</version>
         </bundle>
         <bundle>
            <name>org.eclipse.kura.protocol.pcn</name>
            <version>1.0.0.SNAPSHOT</version>
         </bundle>
         <bundle>
            <name>javax.usb.common</name>
            <version>1.0.2</version>
         </bundle>
         <bundle>
            <name>org.eclipse.kura.core</name>
            <version>1.0.0.SNAPSHOT</version>
         </bundle>
         <bundle>
            <name>org.eclipse.kura.api</name>
            <version>1.0.0.SNAPSHOT</version>
         </bundle>
         <bundle>
            <name>org.eclipse.kura.web</name>
            <version>1.0.0.SNAPSHOT</version>
```

```
        </bundle>
        <bundle>
          <name>javax.comm</name>
          <version>2.2.0</version>
        </bundle>
        <bundle>
          <name>org.eclipse.paho.mqtt-client</name>
          <version>1.0.1.SNAPSHOT</version>
        </bundle>
        <bundle>
          <name>edc-client</name>
          <version>2.1.0.SNAPSHOT</version>
        </bundle>
      </bundles>
    </package>
</packages>
```

The deployment package XML message is comprised of the following package elements:

- Symbolic name
- Version
- Bundles that are managed by the deployment package along with their symbolic name and version

The "DEPLOY-V1" application supports the *read*, *start/stop*, and *install/uninstall* resource operations as described in the following sections.

Read All Deployment Packages

This operation provides the deployment packages installed in the OSGi framework.

- Request Topic:
  - **$EDC/account_name/client_id/DEPLOY-V1/GET/packages**

- Request Payload:
  - Nothing application-specific beyond the request ID and requester client ID

- Response Payload:
  - Installed deployment packages serialized in XML format

Install a Deployment Package

This operation installs a deployment package in the OSGi framework.

- Request Topic:
  - **$EDC/account_name/client_id/DEPLOY-V1/EXEC/install**

- Request Payload:

    - The following application-specific properties in addition to the request ID and requester client ID:
        - A **deploy.url** property that provides the URL to be used by the receiving application to download the deployment package.
        - Alternatively, the deployment package is present in a body property of the MQTT payload. The **deploy.filename** property provides the filename of the deployment package on the receiving device.

- Response Payload:
    - **deploy.pkg.name** provides the symbolic name of the deployment package
    - **deploy.pkg.version** provides the version of the deployment package

Uninstall a Deployment Package

This operation uninstalls a deployment package.

- Request Topic:
    - **$EDC/account_name/client_id/DEPLOY-V1/EXEC/uninstall**

- Request Payload:
    - **deploy.pkg.name** provides the symbolic name of the deployment package

- Response Payload:
    - Nothing application-specific beyond the response code

Read All Bundles

This operation provides all the bundles installed in the OSGi framework.

- Request Topic:
    - **$EDC/account_name/client_id/DEPLOY-V1/GET/bundles**

- Request Payload:
    - Nothing application-specific beyond the request ID and requester client ID

- Response Payload:
    - Installed bundles serialized in XML format

The following XML message is an example of a bundle:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bundles>
   <bundle>
```

```
    <name>org.eclipse.osgi</name>
    <version>3.8.1.v20120830-144521</version>
    <id>0</id>
    <state>ACTIVE</state>
  </bundle>
  <bundle>
    <name>org.eclipse.equinox.cm</name>
    <version>1.0.400.v20120522-1841</version>
    <id>1</id>
    <state>ACTIVE</state>
  </bundle>
</bundles>
```

The bundle XML message is comprised of the following bundle elements:

- Symbolic name
- Version
- ID
- State

Start a Bundle

This operation starts a bundle identified by its ID.

- Request Topic:
  - **$EDC/account_name/client_id/DEPLOY-V1/EXEC/start/bundle_id**

- Request Payload:
  - Nothing application-specific beyond the request ID and requester client ID

- Response Payload:
  - Nothing application-specific beyond the response code

Stop a Bundle

This operation stops a bundle identified by its ID.

- Request Topic:
  - **$EDC/account_name/client_id/DEPLOY-V1/EXEC/stop/bundle_id**

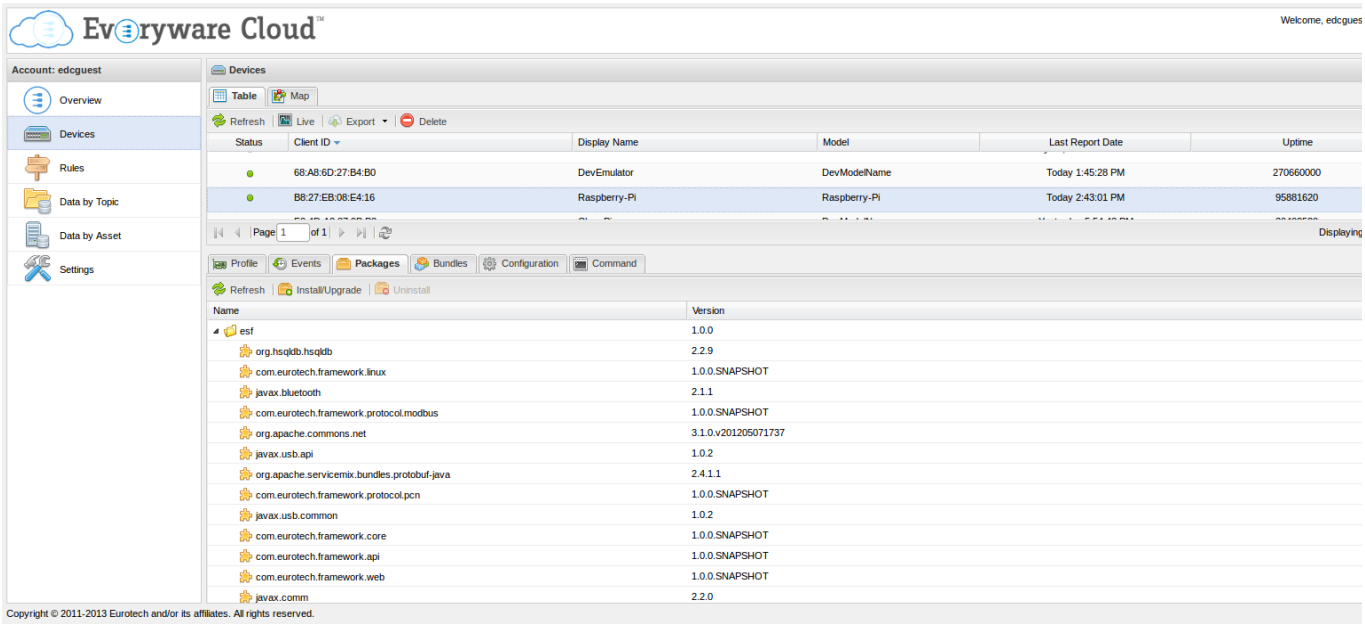
- Request Payload:
  - Nothing application-specific beyond the request ID and requester client ID

- Response Payload:
  - Nothing application-specific beyond the response code

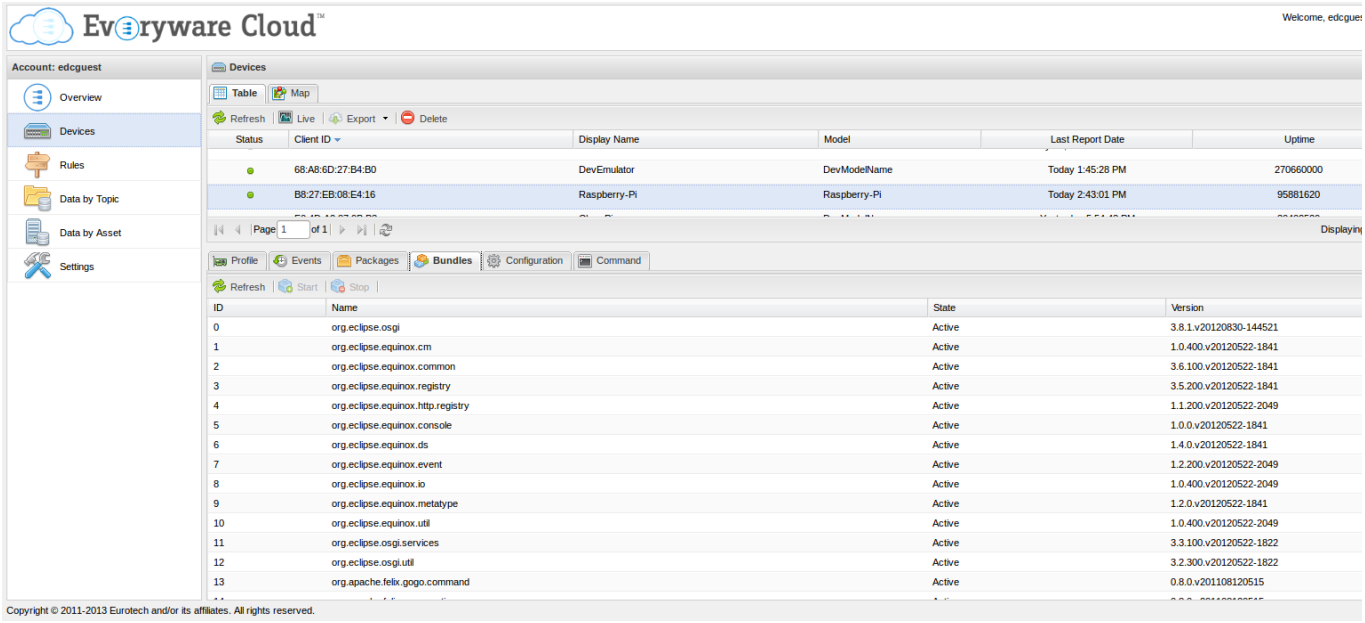
Example Management Web Application

The previously described read, start/stop, and install/uninstall resource operations can be leveraged to develop a web application that allows for remote package deployment via MQTT though a web user-interface.

The screen capture that follows shows an example administration application where, for a given IOT gateway, a list of all deployed packages is presented to the administrator. It also provides the ability to install/uninstall or update the packages.



The **Bundles** tab lists all installed bundles and provides the ability to start and stop them as shown in the screen capture that follows.


Everyware Cloud™
Welcome, edcgues
Account: edcguest
Overview
Devices
Rules
Data by Topic
Data by Asset
Settings
Devices
Table
Map
Refresh
Live
Export
Delete
Status
Client ID
Display Name
Model
Last Report Date
Uptime
68:A8:6D:27:B4:B0
DevEmulator
DevModelName
Today 1:45:28 PM
270660000
B8:27:EB:08:E4:16
Raspberry-Pi
Raspberry-Pi
Today 2:43:01 PM
95881620
Page 1 of 1
Displaying
Profile
Events
Packages
Bundles
Configuration
Command
Refresh
Start
Stop
ID
Name
State
Version
0 org.eclipse.osgi Active 3.8.1.v20120830-144521
1 org.eclipse.equinox.cm Active 1.0.400.v20120522-1841
2 org.eclipse.equinox.common Active 3.6.100.v20120522-1841
3 org.eclipse.equinox.registry Active 3.5.200.v20120522-1841
4 org.eclipse.equinox.http.registry Active 1.1.200.v20120522-2049
5 org.eclipse.equinox.console Active 1.0.0.v20120522-1841
6 org.eclipse.equinox.ds Active 1.4.0.v20120522-1841
7 org.eclipse.equinox.event Active 1.2.200.v20120522-2049
8 org.eclipse.equinox.io Active 1.0.400.v20120522-2049
9 org.eclipse.equinox.metatype Active 1.2.0.v20120522-1841
10 org.eclipse.equinox.util Active 1.0.400.v20120522-2049
11 org.eclipse.osgi.services Active 3.3.100.v20120522-1822
12 org.eclipse.osgi.util Active 3.2.300.v20120522-1822
13 org.apache.felix.gogo.command Active 0.8.0.v20110120515
Copyright © 2011-2013 Eurotech and/or its affiliates. All rights reserved.