MQTT Protocol Adapter

Draft

11 Pages

Abstract

In the IoT domain there is a widespread of communication protocols available for letting devices interact with eachother. A popular publish-subscribe protocol for communicating with IoT devices is MQTT. This RFC focuses on ways to integrate the MQTT protocol with OSGi. The goal to bridge MQTT to the existing OSGi EventAdmin.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 10.1.

```
Source code is shown in this typeface.
```

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	*September 5 2016*	*Initial contribution - Tim Verbelen*
0.1	*December 2016*	*Updates after discussion at Hursley F2F – Tim Verbelen*
0.2	*January 2017*	*API Updates after IoT EG conf call feedback – Tim Verbelen*

1 Introduction

Internet of Things (IoT) is becoming an important application domain of OSGi. The ability to run an OSGi framework on a gateway device as well as a Cloud server, together with the ability of transparently calling remote services using distributed OSGi makes it perfect base for an IoT platform. However, the proliferation of IoT protocols makes it difficult to integrate the many technologies available. One popular IoT protocol for connecting such IoT devices is MQTT. This RFC provides a solution to transparently handle MQTT events in OSGi via the EventAdmin.

2 Application Domain

MQTT is a lightweight publish/subscribe messaging protocol on top of TCP/IP standardized by OASIS [3]. It is widely used in IoT applications. MQTT uses a central broker where all clients send their messages to and where clients can subscribe on certain topics.

MQTT topics are represented as hierarchically structured strings with a forward slash topic separator, i.e. “sensors/temperature/mysensor1”. To subscribe to topics, filters can be used containing wildcard characters. The hash sign '#' is used as a multi-level topic wildcard, the plus sign '+' is used as a single level wildcard. Topic names starting with a dollar sign '$' are special topics that relate to the event broker, and clients should not use topic names starting with '$' to exchange messages with other clients.

The MQTT broker supports multiple QoS levels: QoS level 0 : best effort delivery: the message is not acknowledged by the receiver or stored and redelivered by the sender. QoS level 1: the message is guaranteed to be delivered at least once to the receiver. The sender stores the messages until it gets an acknowledgement. QoS level 2: the message is delivered exactly once to the receiver. This is the highest and slowest QoS level.

MQTT also supports so called “last will and testament” messages, which are sent out by the broker when the sender loses connectivity with the broker. The MQTT specification does not specify any format the message payload should adhere to. However, the application developer will know the payload format, as this is implicitly defined by the topic(s) he subscribes to.

3 Problem Description

Although there are already solutions for using MQTT within an OSGi application, these still leave it up to the developer to use the specific APIs of the MQTT library used, which is currently not standardized in OSGi. For example the Eclipse Paho [4] MQTT client API is used in the Eclipse Kura [5] platform. To facilitate interaction with an MQTT broker, this RFP aims to provide a service API for interacting with an MQTT broker.

4 Requirements

- M0011 – The solution MUST define an service API to publish and subscribe to an MQTT broker
- M0040 – The solution MUST be configurable (i.e. which MQTT broker to connect to)

- M0050 – The solution MUST support the different MQTT QoS levels
- M0060 – The solution MUST support providing last will and testament messages
- M0070 – The solution SHOULD provide a builder to create a new connection to an MQTT broker

5 Technical Solution

5.1 MQTT Service

In case you want to directly communicate with the MQTT broker instead of using EventAdmin, an MQTTService can be used:

```
public interface MQTTService {

	public PushStream<ByteBuffer> subscribe(String topic);

	public PushStream<ByteBuffer> subscribe(String topic, Qos qos);

	public void publish(String topic, ByteBuffer data) throws Exception;

	public void publish(String topic, ByteBuffer data, Qos qos) throws Exception;

	public void publishRetained(String topic, ByteBuffer data, Qos qos) throws
Exception;
}
```

Subscribe will return you a PushStream with a ByteBuffer with payload data of the MQTT events. Publish will publish an MQTT message to a certain topic with data payload. The ByteBuffer object must be backed by an accessible byte array. When the pushstream is closed, the client is unsubscribed from the topic.

The Object Convertor can be used to then convert the payload to whatever object by mapping the stream. For example in case the payload is a json string that can be mapped to a DTO:

```
mqtt.subscribe("osgi/trains/observation")
	.map(buffer -> converter.convert(new
String(buffer.array())).to(Observation.class))
	.forEach(o -> System.out.println("Observation: "+o));
```

5.2 MQTT QoS

MQTT has three QoS levels: 0) deliver at most once 1) deliver at least once and 2) deliver exactly once. The QoS can be indicated on the publish method (QoS used to publish message from client to broker) as well as on the subscribe method (QoS used for receiving messages from the broker).

The default QoS is 1.

In the MQTTService methods we use a Qos enum type for setting the QoS:

```
public enum Qos {
    AT_MOST_ONCE,
    AT_LEAST_ONCE,
    EXACTLY_ONCE
}
```

5.3 Retained messages

When publishing a message can be marked as retained. The broker will store the last retained message and corresponding QoS for that topic. When a client subscribes to a topic that matches the topic of a retained message, it will immediately receive this retained message.

To publish a retained message, use the publishRetained method of the MQTTService

5.4 Setting up a broker connection

An MQTT service provider bundle can register an MQTTService instance into the service registry that can be used by any bundle in the framework to publish/subscribe to MQTT topics. However, often a bundle would like his very own MQTTService instance, i.e. providing authentication or other options:

```
public interface MQTTServiceBuilder {

      // create the actual MQTTService
      public MQTTService create();

      // connect to a broker
      public MQTTServiceBuilder connect(String serverURI, String clientId);

      // clean the session when the client reconnects
      public MQTTServiceBuilder clean();

      // automatically reconnect
      public MQTTServiceBuilder autoReconnect();

      // set max inflight
      public MQTTServiceBuilder maxInFlight(int max);

      // set connection timeout
      public MQTTServiceBuilder timeout(Duration timeout);

      // set keepAlive interval
      public MQTTServiceBuilder keepAlive(Duration keepAlive);

      // set username
      public MQTTServiceBuilder username(String username);

      // set password
      public MQTTServiceBuilder password(String password);

      // set ssl properties
      public MQTTServiceBuilder ssl(Properties props);

      // set last will
```

```
    public MQTTServiceBuilder lastWill(String topic, ByteBuffer data, Qos qos,
boolean retained);

}
```

5.5 Last will and testament

The last will message can be provided with the MQTTServiceBuilder when creating an MQTTService.

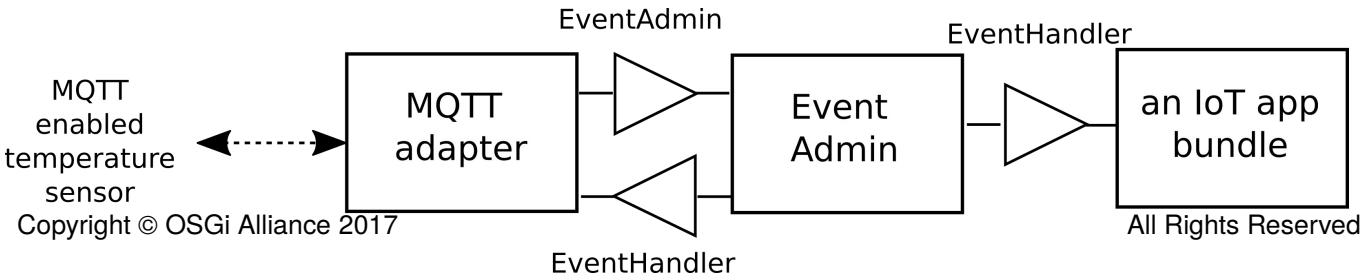
```
MQTTServiceBuilder builder = …;
MQTTService mqtt = builder.connect(brokerURI, clientId).lastWill(“/will/topic”,
payload, 1, false).create();
```

6 Data Transfer Objects

7 Javadoc

8 Considered Alternatives

A considered alternative solution to integrate MQTT with OSGi is by providing an adapter bundle that listens for MQTT messages on given topics, convert these to OSGi Events and publishes them out to EventAdmin, and similarly converts OSGi Events to MQTT messages that are sent back to the MQTT broker.



8.1 Configuring the MQTT adapter

The MQTT adapter has to be configured with the MQTT broker to connect to, optionally authentication username/password, and which topics to subscribe/publish to. MQTT topic names support more wildcard options that OSGi EventAdmin, therefore the use of wildcards is restricted to a trailing '#' in an MQTT topic, which maps to a trialing '*' in the corresponding EventAdmin topic.

The MQTT adapter is configured with the topics it should listen to from OSGi EventAdmin that should be broadcasted to MQTT, and which topics it should subscribe to with the MQTT broker that are to be published via OSGi EventAdmin.

An example minimal configuration could be:

```
{
   "service.factoryPid":"org.osgi.mqtt",
   "broker" : "tcp://iot.eclipse.org:1883",
   "event.topics" : ["my/topic","my/wildcarded/topic/*"]
   "mqtt.topics" : ["mqtt/topic"]
}
```

8.2 Converting payload

MQTT messages have no specified payload format. OSGi events on the other hand are formatted as a key-value map. The following conversion rules are used:

- The OSGi event map is converted to a json string as MQTT payload using the Object Convertor.
- An MQTT payload that is a json string is converted to an OSGi event map using the Object Convertor
- If the MQTT payload cannot be converted, the resulting OSGi event map has a single key “payload” with the raw byte[] from the MQTT message.

8.3 Avoiding message publishing loops

When there is bidirectional traffic (both events published and received) on a single topic, this might result in an endless loop (the adapter publishes the event to MQTT, receives it again from the MQTT broker as it is also subcriber, sends it out to EventAdmin again …). Therefore, the adapter should add an extra “sender” field to the MQTT messages it sends out, containing the framework UUID. Messages containing the framework UUID as sender should not be propagated further by this MQTT adapter.

8.4 Whiteboard pattern

Other bundles can also set the 'mqtt.topics' property on any service it registers. This way a bundle can request to also publish events on these topics to an MQTT broker. Similarly an EventHandler with 'event.topics' property can set 'mqtt.subscribe=true' to also subscribe on these topics via the MQTT adapter.

This approach was discarded in favor of a MQTT Service approach, for the following reasons:

- with a Pushstream-based MQTT Service API, it becomes trivial to map MQTT event streams to EventAdmin, if this behavior is desired

- mapping OSGi events to MQTT payload can easily be done using the Object Convertor spec, mapping MQTT event to OSGi events is highly dependent on the internal format of the 3rd party MQTT provider, hence no much value in this

9 Security Considerations

10 Document Support

10.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 29 September 2014. OASIS Standard. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[4]. Eclipse Paho, https://www.eclipse.org/paho/

[5]. Eclipse Kura, https://www.eclipse.org/kura/

10.2 Author's Address

Name	Tim Verbelen
Company	iMinds, Ghent University
Address	
Voice	
e-mail	tim.verbelen@intec.ugent.be

10.3 Acronyms and Abbreviations

10.4 End of Document