RFP 197 - Type Safe Events

Draft

9 Pages

Abstract

10 point Arial Centered.

This RFC aims to gather requirements for updating, or superseding, the OSGi Event Admin specification. Specific areas of improvement include the type safety of event data, monitoring, and tracking of undelivered events. These features are necessary enhancements if the Event Admin pattern is to remain used by modern applications in the future.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

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 6.1.

```
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	Jun 18 2019	Initial version of the document Tim Ward, Paremus tim.ward@paremus.com
0.1	Jul 5 2019	Incorporating feedback from the CPEG and EEG calls Tim Ward, Paremus tim.ward@paremus.com

1 Introduction

The OSGi Event Admin specification is one of the earliest specifications defined by the Compendium. It provides a useful, flexible model for exchanging events between modules. The design and usage of Event Admin, however, shows evidence of the specification's age.

- Events are sent and received as opaque maps of key-value pairs. The "schema" of an event is therefore ill-defined and relies on "magic strings" being used correctly
- Events that are sent but have no Event Handlers are silently discarded with no way to know that the event went unhandled
- There is no simple way to monitor the flow of events through the system

The BRAIN-IoT Horizon 2020 project[3]. is an example of a modern OSGi application that could have used Event Admin, except for the above issues.

2 Application Domain

Eventing systems are a common part of software programs, used to distribute information between parts of an application.

2.1 Event Admin in OSGi

The standard OSGi Event Admin listener pattern requires that event handlers are registered in the OSGi service registry. These services are called by Event Admin whenever an appropriate Event is delivered using the Event Admin service. The Event Handler API is not type safe, in that it receives an Event containing String keys mapped to Object values.

Similarly, an Event Source must correctly construct an Event from String keys and Object values, then sent out with a named topic. This can lead to problems if more than one Event Source sends to the same topic, as they may differ slightly in the keys and value types that they use.

If an Event Source sends an event then there is no feedback about whether any Event Handler received the Event. Furthermore there is no way to determine what events are being sent. Systems using Event Admin can therefore end up with failure modes which are very difficult to diagnose.

2.2 Terminology + Abbreviations

- **Event** – A set of data created by an Event Source, encapsulated as an object and delivered to one or more Event Handers
- **Event Topic** – A String identifying the “topic” of an Event, effectively defining the schema and purpose of the event
- **Event Source** – A software component which creates and sends events
- **Event Handler** – A software component which receives events
- **DTO** – A Data Transfer Object as per the OSGi DTO Specification
- Event Bus – A software component used by an Event Source and responsible for delivering Events to Event Handers. For example The OSGi Event Admin service.

3 Problem Description

The Event Admin Specification exists to solve the issue of Eventing within an OSGi framework, so why is it now insufficient?

3.1 Event Schemas and Type Safety

One of the primary problems with Event Admin is the inability to reliably and safely consume Events. To understand the data in an Event the Event Handler must defensively check for the existence of property keys, and then for the type of the value associated with a given key. This is because there is no concept of “schema” or “contract” for an Event Topic and the messages are untyped, so each participant has to continually work out what kind of message it has received, validate it, handle errors and missing info, work out what it should send in response.

Use of the OSGi DTO and Converter specifications can improve this model, however it significantly increases the amount of boilerplate needed to write both an Event Source and an Event Handler.

Using “schemaless” events is fine if we don't want to go to the trouble of defining a contract for a particular interaction, but the risk is that modules become *more* tightly coupled because of hidden assumptions about the form of events they exchange.

3.2 Event Monitoring

The current Event Admin only specifies how to send and receive events, but not how to monitor the flow of events in the system. The best that can be achieved is to register an Event Handler which listens to all Event Topics, however this does not allow for easy filtering of data, nor does it provide information about the source of the event. Tools that wish to analyze the flow of event data through the system are therefore unable to simply do so.

3.3 Unhandled Events

If an event is sent by an Event Source it is typically expected that there will be at least one listener for the event. If there are no listeners then the current Event Admin Behaviour is to silently discard the event. In many systems the correct response to an unhandled event is to halt processing, or at least to warn a user/operator of the unhandled event.

4 Use Cases

The following use cases are predominantly from the BRAIN-IoT project:

4.1 Type Safe Eventing

As a user I wish to produce an Event Handler which consumes events produced by an IoT device. The events have a defined schema and I wish to simply and easily process the event data without having to extensively validate it. The events should therefore be delivered in a type-safe format. This enables me to write readable, maintainable code with simple logic.

4.2 Operator Monitoring

As a systems administrator I wish to inspect a runtime system to determine that it is operating correctly. This involves viewing the events coming from IoT sensors, and the aggregate statistics events that are published based on the sensor data. This enables me to verify the operation of my system at runtime

4.3 Dynamic Runtime Assembly

As a smart systems operator I wish my runtime to detect when new sensors (event sources) have been added that are producing events which are otherwise unhandled. This in turn can notify the operator to install (or automatically install) an Event Handler capable of processing the new Event data.

4.4 Generic Event Handlers

As a developer I wish to produce a generically configurable event handler which is capable of consuming events, performing some intermediate processing on them, and then publishing on an event based on the processing.

For example:

- I wish to add a “rolling average” function which consumes events from a noisy sensor, and republishes an event containing the average of the last N values. This is then consumed by users interested in the “smoothed” data
- I wish to add a “debounce” function which suppresses events from a detector for a short period of time after it has been triggered. This helps to prevent multiple reactions to the same physical event. The Event Handler therefore consumes and republishes events, but has a “quiet period” after receiving an event during which events are not republished.

4.5 Listener specific event views

As the developer of an Event Handler I wish to consume event data relevant to my handler, without necessarily providing access to the full set of data in the event. For example a “smart weather station” may provide regular events containing air temperature, barometric pressure, humidity, wind speed and direction etc. If my Event Handler only wishes to react to temperature readings then requiring it to mirror the entire schema is wasteful and confusing.

It should also be noted that there may be multiple Event Handlers interested in different subsets of the data, for example another handler may be interested in wind speeds so that it can optimize the energy output of a wind farm, and/or shut down turbines if the wind speed gets too high. These different Event Handlers should be able to consume events from the same Event Source, while using different type-safe data objects relevant to their domains of interest.

4.6 Event Schema versioning

The schema for an event will change over time as the software (and potentially hardware) is upgraded and enhanced. It must be possible for an Event Handler to safely consume (or filter out) events produced at higher or lower version than the consumer is wired to.

For example if as a developer I wish to consume data from an Event Source which was only added to the schema in version 1.1 then I must be able to reliably filter out events sent using version 1.0 of the schema, but accept events sent using version 1.2 of the schema.

It may also be necessary for an Event Handler to be able to consume events from an older version of the schema than they are wired to, knowing that some data fields may not be populated.

5 Requirements

TSE-010 – The solution MUST enable Event Sources and Event Handlers to work with Type Safe Event objects without requiring the use of an intermediate Map object in ~~client~~application code

TSE-020 – The solution SHOULD allow the use of Map structures in Event Handlers and Event Sources to cope with “reflective” operations such as rolling average and debouncing.

TSE-030 – The solution MUST provide a way for an operator to monitor the events being sent by Event Sources

TSE-040 – The solution MUST provide a way for a bundle to be notified when there are no suitable Event Handlers to process an Event

TSE-050 – The solution MUST allow an Event Handler to consume an Event as a Type Safe Event object which is different from the Type Safe Event Object produced by the Event Source. The Event Handler’s Type Safe Object MAY be required to be a partial match for the Event Source’s Type Safe Object, i.e. the Event Bus is not necessarily required to perform schema transformations such as changing field names.

TSE-060 – The solution SHOULD allow an Event Handler to declare a minimum version for the schema of events that it consumes. The aim of this requirement is to prevent errors if two Event Sources deliver events to the same topic using different schema versions.

6 Document Support

6.1 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]. The BRAIN-IoT Horizon 2020 project - http://www.brain-iot.eu/

6.2 Author's Address

Name	Tim Ward
Company	Paremus
Address	
Voice	
e-mail	tim.ward@paremus.com

6.3 End of Document