RFC-244 Type Safe Eventing

Draft

11 Pages

Abstract

This RFC aims to update, or supersede, the OSGi Event Admin specification to make the specification more reliable and natural for application code to use. Specific areas of improvement include the type safety of event data, monitoring of event flow, 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 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	Sep 16 2019	Initial Revision 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.

The limitations of Event Admin are made even more obvious by the OSGi R7 release, which includes specifications for Data conversion (using the OSGi converter[4].) and stream processing for typed data objects (using PushStreams[5].). The simplicity and developer-friendly APIs provided by these models provide a reasonable goal for the future usability of any solution proposed by this RFP.

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 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 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.

5 Technical Solution

The following technical solution is proposed to extend the Event Admin Model with the necessary features

5.1 API Separation

The Type Safe Eventing Service will, by necessity, be a significant change from the existing Event Admin API. The proposed changes will therefore be targeted at a different package (`org.osgi.service.event.typed`), rather than the existing Event Admin package (`org.osgi.service.event`).

5.2 Sending Events

Event Sources are able to send events using the Type Safe Eventing Service, which is registered as a service in the OSGi service registry. Event delivery is always asynchronous, meaning that there is no guarantee that any Event Handlers have received the event data before control is returned to the Event Source.

5.2.1 Event Topics

As with OSGi's Event Admin, Events are sent to a topic. This topic defines both the Event's schema, and also provides a coarse scale hierarchy for the event data.

5.2.2 Sending Typed Events

Typed event data is sent using on of the two `deliver` methods:

1) The first deliver method takes a `String` topic name and an `Object` event.

2) The second deliver method takes only an Object. The topic name is set to the fully qualified class name of the event class

The first method is familiar to those who have used the OSGi Event Admin service, but the second will be unfamiliar as it has no corresponding method. In the case where there is no meaningful topic hierarchy, and/or there is no reuse of the Event Schema across multiple topics then creating an arbitrary topic name is simply an opaque string value that must be carefully copied or used as a constant. In this eventuality the fully qualified class name of the Event object provides a suitable topic name which cannot be accidentally mistyped

5.2.3 Sending Untyped Events

Untyped Event Data is also important as some Event consumers will wish to operate on the event data without necessarily having compile-time access to the Event Class Definition. At this point the Event data must be gathered in a map data structure with `String` keys mirroring the Event schema defined by the DTO that would be used to send a Typed Event. The `Map` is then sent using the deliverUntyped method which takes a `String` topic name and the `Map` containing the event data.

It is obviously not possible for untyped events to be sent without supplying a topic name as there is no sensible default to provide which defines the schema.

5.3 Event Structure

Event objects are expected to conform to OSGi DTO[5]. rules. All methods, all static fields, and any non public instance fields must be ignored by the Type Safe Eventing Service when processing the Event data.

5.3.1 Nested Data Structures

The OSGi Event Admin specification recommends (although does not prohibit) that events do not contain nested data structures more complex than Lists or Sets. The Type Safe Event Service is different, in that DTO-type events may contain arbitrarily deep data structures. The only requirement is that the DTO structure contains no cycles.

When sending an untyped event with a nested structure each nested DTO value should be provided as another Map with string keys, and so on, until the leaf data is reached.

5.4 Receiving Events

Receiving an Event is performed by registering an appropriate Event Handler Service in the Service Registry. Events are then delivered using the whiteboard pattern.

5.4.1 Receiving Typed Events

Typed Events are received by registering a `TypedEventHandler` implementation. This service has a single method `notify` which receives the `String` topic name and `Object` event data. The TypedEventHandler implementation must be registered as a service in the service registry using the TypedEventHandler interface.

The TypedEventHandler interface is parameterized, and so it is expected that the implementation reifies the type parameter into a specific type. In this case the Type Safe Event Service must adapt the Event object into the type defined by the TypedEventHandler implementation.

If the TypedEventHandler implementation is unable to reify the type, or the required type is more specific than the reified type, then the Typed Event Handler must be registered with the `event.type` property. This property has a string value containing the fully-qualified type name of the type that the Event Handler expects to receive. This type must be loaded from the bundle which registered the Event Handler service, and used as the target type when converting events.

By default the reified type of the TypedEventHandler will be used as the target topic for the Event Handler. If the `event.type` property is set then this is used as the default instead of the reified type. To use a different topic the Event Handler service may register the service with an `event.topics` property specifying the topic(s) as a String+ value.

5.4.2 Receiving Untyped Events

Untyped Events are received by registering an UntypedEventHandler implementation. This service has a single method `notifyUntyped` which receives the `String` topic name and `Map` event data. The UntypedEventHandler implementation must be registered as a service in the service registry using the UntypedEventHandler interface.

When delivering an event to an UntypedEventHandler the Typed Event Service must convert the event data to a nested map structure.

The `event.topics` property must be used when registering an UntypedEventHander service. If it is not then no events will be delivered to the Event Handler service

5.5 Monitoring Events

TODO

5.6 Unhandled Events

TODO

6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

7 Javadoc

Please include Javadoc of any new APIs here, once the design has matured. Instructions on how to export Javadoc for inclusion in the RFC can be found here: https://www.osgi.org/members/RFC/Javadoc

8 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

9 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

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

[4]. OSGi Converter Specification - https://osgi.org/specification/osgi.cmpn/7.0.0/util.converter.html

[5]. OSGi PushStream Specification - https://osgi.org/specification/osgi.cmpn/7.0.0/util.pushstream.html

[6]. OSGi DTO Specification - https://osgi.org/specification/osgi.core/7.0.0/framework.dto.html

10.2 Author's Address

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

10.3 Acronyms and Abbreviations

10.4 End of Document