



OSGiTM
Alliance

RFC-244 Type Safe Eventing

Draft

31 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

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 Document Information.....	2
0.1 License.....	2
0.2 Trademarks.....	3
0.3 Feedback.....	3
0.4 Table of Contents.....	3
0.5 Terminology and Document Conventions.....	4
0.6 Revision History.....	4
 1 Introduction.....	 4
2 Application Domain.....	5
3 Problem Description.....	5
4 Requirements.....	5
5 Technical Solution.....	5
6 Data Transfer Objects.....	6
7 Javadoc.....	6
8 Considered Alternatives.....	6

9 Security Considerations.....7**10 Document Support.....7**

10.1 References.....7

10.2 Author's Address.....7

10.3 Acronyms and Abbreviations.....7

10.4 End of Document.....7

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
<u>0.1</u>	<u>Sep 19 2019</u>	<u>Updates from the Sofia F2F meeting, additions to monitoring and unhandled events</u>

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 Handlers
- **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 Handlers. 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.type`), rather than the existing Event Admin package (`org.osgi.service.event`).

5.2 Event Structure

Type Safe 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.

5.2.1 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.

5.2.2 Non-standard Event Types

Some Event schemas may be represented by an existing type which does not match the OSGi DTO rules. In this case there are two main options

1. Create a DTO representation of the event schema, and convert from the existing type in code
2. Create a custom converter capable of transforming the event data

5.3 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 using the TypeSafeEventBus interface. 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.

5.3.1 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.

5.3.2 Sending Typed Events

Typed event data is sent using one 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

5.3.3 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.

5.3.4 Sending Custom Event Data

Custom Event Data is Event Data which does not fit OSGi DTO rules, and therefore cannot be automatically type converted by the Event Bus. If Custom Event Data must be sent by the Event Source then the `deliverCustom`

method, which takes a `String` topic, `Object` event data and a `Converter` suitable for converting the `Event Data` into:

1. A nested Map structure, suitable for use by Untyped Event Handlers
2. A DTO structure, suitable for use in “standard” views of the Event Schema
3. Any other Data Structures which are supported ways to receive the custom Event Data type.

5.3.5 Error Handling

It is not possible to know that an Event cannot be delivered until delivery is attempted. It is therefore not possible (or acceptable, given the asynchronous nature of delivery) to throw an exception to the sender of an event if the event cannot be converted into the relevant target type

5.4 Event Topics

Each Event is sent to a topic. The topic defines the schema of an event, and potentially context about the event. The Event Topic syntax is a String following bundle symbolic name rules

5.4.1 Topic Hierarchies

Topics can be made hierarchical by adding a sub-topic to an existing topic name, using a ``/`` as a separator character. A sub-topic is distinct from its parent topic, and may have different rules, however it is expected that the schema of a sub-topic is compatible with the schema of the parent topic (i.e. that an Event Handler can consume events from a sub-topic using the schema of the parent topic). Therefore sub-topic schemas may contain additional data, but should not remove data from the schema of the parent topic.

5.5 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.

5.5.1 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.

5.5.2 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 `UntypedEventHandler` service. If it is not then no events will be delivered to the Event Handler service

5.5.3 Wildcard Topics

The `event.topics` property may contain one or more **wildcard** topics. These are Strings which contain a topic name and append `/*`. This value means that the Event Handler must be called for Events in the named topic and all sub-topics

5.5.4 Event Filtering

TODO – is this needed?

5.5.5 Error Handling

There are several possible error scenarios for Event Handlers:

- `TypedEventHandler`, Event Type not discoverable – in this case there is no reified type information, nor is there an `event.type` property, and therefore the target type for the event is not known. In this situation there is no way for the Event Bus to correctly target an event schema, and the Event Handler must be ignored.
- `TypedEventHandler`, Event type discoverable but not loadable – in this case the type information is discoverable (either via the reified type or the `event.type` property) but the classloader of the bundle which registered the service is unable to load it. In this situation there is no way for the Event Bus to correctly target an event schema, and the Event Handler must be ignored.
- All Handler Types, Event cannot be converted to the target type – in this case a specific Event cannot be submitted to the Handler. If this error occurs repeatedly then the Event Bus may choose to ignore the Event Handler service
- All Handler Types, Invalid `event.topic` String – in this case one or more Event Topics declared in the `event.topics` property is syntactically invalid. In this situation the Event Handler must be ignored.

5.6 Monitoring Events

An important part of a software system is the ability to monitor it appropriately to determine whether it is functioning correctly, without having the measurements disrupt the system. To this end the Type Safe Event Specification defines a `TypeSafeEventMonitor` service which can be used to monitor the flow of events through the Event Bus.

5.6.1 Monitoring the Events

Events flowing through the Event Bus can be monitored using one of the `monitorEvents` methods from the `TypeSafeEventMonitor` service. These methods return a `PushStream` which delivers `MonitorEvent`

instances each time an event is sent via the `TypeSafeEventBus`. The monitor events contain the event topic, the event data, and a timestamp indicating when the event was sent

5.6.2 Event History

In a running system it is often useful to be able to connect monitoring to replay recent data immediately after a problem has occurred. For that reason `TypeSafeEventManager` instances may store past events so that they can be replayed if requested. There are two `monitorEvents` methods capable of replaying history

- The first method takes an int representing the number of past events that should be replayed from the cached history
- The second method takes an `Instant`, representing the time from which monitoring events should start.

Note that storing Event History is considered a best-effort option and it is not required that the implementation supply the full set of requested events. If insufficient past events are available then the implementation must provide the maximum amount of history available.

When a stored event is discarded then the oldest event must be discarded before any newer events. This avoids a monitor seeing gaps in the history that they do receive.

5.7 Unhandled Events

Unhandled Events are events sent by an Event Source but which have no registered Event Handler in the service registry. Rather than these events being discarded, the `TypeSafeEventBus` will search the registry for services implementing `UnhandledEventHandler`.

If any services are found then the `TypeSafeEventBus` will call the `notifyUnhandled` method passing the topic and event data to the `UnhandledEventHandler`.

5.8 Runtime Service

TODO – other whiteboard specs (e.g. the HTTP/JAX-RS whiteboards) have a Runtime Service, should this specification?

5.9 Event Filtering

TODO – Do we want to support filtering?

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

OSGi Javadoc

9/19/19 6:42 PM

Package Summary		Page
org.osgi.service.event.type	Type Safe Event Package Version 1.0.	14
org.osgi.service.event.type.annotations	Type Safe Event Annotations Package Version 1.0.	21
org.osgi.service.event.type.monitor	Type Safe Event Monitoring Package Version 1.0.	23
org.osgi.service.event.type.propertytypes	Type Safe Event Component Property Types Package Version 1.0.	28

Package org.osgi.service.event.typed

@org.osgi.annotation.versioning.Version(value="1.0.0")

Type Safe Event Package Version 1.0.

See:

Description

Interface Summary		Page
EventConstants	Defines standard names for Type Safe Event properties.	15
TypedEventHandler	Listener for Typed Events.	16
TypeSafeEventBus	The Type Safe Event service.	17
UnhandledEventHandler	Listener for Unhandled Events.	19
UntypedEventHandler	Listener for Untyped Events.	20

Package org.osgi.service.event.typed Description

Type Safe Event Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

Import-Package: org.osgi.service.event.typed; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.event.typed; version="[1.0,1.1)"

Interface EventConstants

org.osgi.service.event.type

@org.osgi.annotation.versioning.ProviderType
public interface EventConstants

Defines standard names for Type Safe Event properties.

Field Summary			Page
String	TYPE_SAFE_EVENT_IMPLEMENTATION	The name of the implementation capability for the Event Admin specification	15
String	TYPE_SAFE_EVENT_SPECIFICATION_VERSION	The version of the implementation capability for the Event Admin specification	15
String	TYPE_SAFE_EVENT_TOPICS	The name of the implementation capability for the Event Admin specification	15
String	TYPE_SAFE_EVENT_TYPE	The name of the implementation capability for the Event Admin specification	15

Field Detail

TYPE_SAFE_EVENT_TYPE

public static final String TYPE_SAFE_EVENT_TYPE = "event.type"

The name of the implementation capability for the Event Admin specification

TYPE_SAFE_EVENT_TOPICS

public static final String TYPE_SAFE_EVENT_TOPICS = "event.topics"

The name of the implementation capability for the Event Admin specification

TYPE_SAFE_EVENT_IMPLEMENTATION

public static final String TYPE_SAFE_EVENT_IMPLEMENTATION = "osgi.event.type"

The name of the implementation capability for the Event Admin specification

TYPE_SAFE_EVENT_SPECIFICATION_VERSION

public static final String TYPE_SAFE_EVENT_SPECIFICATION_VERSION = "1.0.0"

The version of the implementation capability for the Event Admin specification

Interface TypedEventHandler

org.osgi.service.event.typed

Type Parameters:

T - The type of the event to be received

@org.osgi.annotation.versioning.ConsumerType
public interface TypedEventHandler

Listener for Typed Events.

TypedEventHandler objects are registered with the Framework service registry and are notified with an event object when an event is sent.

TypedEventHandler objects are expected to reify the type parameter T with the type of object they wish to receive when implementing this interface. This type can be overridden using the EventConstants.TYPE_SAFE_EVENT_TOPICS service property.

TypedEventHandler objects may be registered with a service property EventConstants.TYPE_SAFE_EVENT_TOPICS whose value is the list of topics in which the event handler is interested.

For example:

```
String[] topics = new String[] {  
    "com/isv/*"  
};  
Hashtable ht = new Hashtable();  
ht.put(EventConstants.TYPE_SAFE_EVENT_TOPICS, topics);  
context.registerService(TypedEventHandler.class, this, ht);
```

ThreadSafe

Method Summary		Page
void	notify(String topic, T event) Called by the TypeSafeEventBus service to notify the listener of an event.	16

Method Detail

notify

```
void notify(String topic,  
            T event)
```

Called by the TypeSafeEventBus service to notify the listener of an event.

Parameters:

- topic - The topic to which the event was sent
- event - The event that occurred.

Interface TypeSafeEventBus

org.osgi.service.event.typed

@org.osgi.annotation.versioning.ProviderType
public interface TypeSafeEventBus

The Type Safe Event service. Bundles wishing to publish events must obtain this service and call one of the event delivery methods.

ThreadSafe

Method Summary		Page
void	deliver (Object event) Initiate asynchronous, ordered delivery of an event.	17
void	deliver (String topic, Object event) Initiate asynchronous, ordered delivery of an event.	17
void	deliverCustom (String topic, Object event, org.osgi.util.converter.Converter converter) Initiate asynchronous, ordered delivery of event data.	18
void	deliverUntyped (String topic, Map<String,?> event) Initiate asynchronous, ordered delivery of event data.	18

Method Detail

deliver

void **deliver**(Object event)

Initiate asynchronous, ordered delivery of an event. This method returns to the caller before delivery of the event is completed. Events are delivered in the order that they are received by this method.

The topic for this event will be automatically set to the fully qualified type name for the supplied event object.

Logically equivalent to calling `deliver(event.getClass().getName(), event)`

Parameters:

`event` - The event to send to all listeners which subscribe to the topic of the event.

deliver

void **deliver**(String topic,
Object event)

Initiate asynchronous, ordered delivery of an event. This method returns to the caller before delivery of the event is completed. Events are delivered in the order that they are received by this method.

Parameters:

`topic` - The topic to which this event should be sent.

`event` - The event to send to all listeners which subscribe to the topic.

deliverUntyped

```
void deliverUntyped(String topic,  
                    Map<String,?> event)
```

Initiate asynchronous, ordered delivery of event data. This method returns to the caller before delivery of the event is completed. Events are delivered in the order that they are received by this method.

Parameters:

topic - The topic to which this event should be sent.

event - A Map representation of the event data to send to all listeners which subscribe to the topic.

deliverCustom

```
void deliverCustom(String topic,  
                  Object event,  
                  org.osgi.util.converter.Converter converter)
```

Initiate asynchronous, ordered delivery of event data. This method returns to the caller before delivery of the event is completed. Events are delivered in the order that they are received by this method.

Parameters:

topic - The topic to which this event should be sent.

event - A Map representation of the event data to send to all listeners which subscribe to the topic.

converter - A converter which can be used to convert the event object into the necessary target types for Event Delivery and Monitoring

Interface UnhandledEventHandler

org.osgi.service.event.typed

@org.osgi.annotation.versioning.ConsumerType
public interface UnhandledEventHandler

Listener for Unhandled Events.

UnhandledEventHandler objects are registered with the Framework service registry and are notified with an event object when an event is sent, but no other handler is found to receive the event

ThreadSafe

Method Summary		Page
void	notifyUnhandled (String topic, Map<String, Object> event) Called by the TypeSafeEventBus service to notify the listener of an unhandled event.	19

Method Detail

notifyUnhandled

void [notifyUnhandled](#)(String topic,
Map<String, Object> event)

Called by the [TypeSafeEventBus](#) service to notify the listener of an unhandled event.

Parameters:

- [topic](#) - The topic to which the event was sent
- [event](#) - The event that occurred.

Interface *UntypedEventHandler*

`org.osgi.service.event.type`

```
@org.osgi.annotation.versioning.ConsumerType
public interface UntypedEventHandler
```

Listener for Untyped Events.

UntypedEventHandler objects are registered with the Framework service registry and are notified with an event object when an event is sent.

UntypedEventHandler objects must be registered with a service property `EventConstants.TYPE_SAFE_EVENT_TOPICS` whose value is the list of topics in which the event handler is interested.

For example:

```
String[] topics = new String[] {
    "com/isv/*"
};
Hashtable ht = new Hashtable();
ht.put(EventConstants.TYPE_SAFE_EVENT_TOPICS, topics);
context.registerService(UntypedEventHandler.class, this, ht);
```

ThreadSafe

Method Summary		Page
void	<code>notifyUntyped</code> (String topic, Map<String, Object> event) Called by the <code>TypeSafeEventBus</code> service to notify the listener of an event.	20

Method Detail

`notifyUntyped`

```
void notifyUntyped(String topic,
                   Map<String, Object> event)
```

Called by the `TypeSafeEventBus` service to notify the listener of an event.

Parameters:

- `topic` - The topic to which the event was sent
- `event` - The event that occurred.

Package org.osgi.service.event.typed.annotations

[@org.osgi.annotation.versioning.Version\(value="1.0.0"\)](#)

[Type Safe Event Annotations Package Version 1.0.](#)

See:

[Description](#)

Annotation Types Summary		Page
RequireTypeSafeEvents	This annotation can be used to require the Event Admin implementation.	22

Package org.osgi.service.event.typed.annotations Description

[Type Safe Event Annotations Package Version 1.0.](#)

[This package contains annotations that can be used to require the Type Safe Event implementation.](#)

[Bundles should not normally need to import this package as the annotations are only used at build-time.](#)

Annotation Type `RequireTypeSafeEvents`

`org.osgi.service.event.typed.annotations`

```
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value={
    ElementType.TYPE,
    ElementType.PACKAGE
})
@org.osgi.annotation.bundle.Requirement(namespace="osgi.implementation",
    name="osgi.event.typed",
    version="1.0.0")
public @interface RequireTypeSafeEvents
```

This annotation can be used to require the Event Admin implementation. It can be used directly, or as a meta-annotation.

This annotation is applied to several of the Event Admin component property type annotations meaning that it does not normally need to be applied to Declarative Services components which use the Event Admin.

Since:

1.4

Package org.osgi.service.event.typed.monitor

@org.osgi.annotation.versioning.Version(value="1.0.0")

Type Safe Event Monitoring Package Version 1.0.

See:

Description

Interface Summary		Page
TypeSafeEventMonitor	The EventMonitor service can be used to monitor the events that are sent using the EventBus, and that are received from remote EventBus instances	26
Class Summary		Page
MonitorEvent	A monitoring event.	24

Package org.osgi.service.event.typed.monitor Description

Type Safe Event Monitoring Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

Import-Package: org.osgi.service.event.typed.monitor; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

Import-Package: org.osgi.service.event.typed.monitor; version="[1.0,1.1)"

Class `MonitorEvent`

`org.osgi.service.event.typed.monitor`

`java.lang.Object`
└─ `org.osgi.dto.DTO`
└─ `org.osgi.service.event.typed.monitor.MonitorEvent`

`@org.osgi.annotation.versioning.ProviderType`
`public class MonitorEvent`
`extends org.osgi.dto.DTO`

A monitoring event.

Field Summary		Page
<code>Map<String, Object></code>	<code>eventData</code> The Data from the Event in Map form	24
<code>Instant</code>	<code>publicationTime</code> The time at which the event was published	24
<code>String</code>	<code>topic</code> The Event Topic	24

Constructor Summary	Page
<code>MonitorEvent()</code>	25

Methods inherited from class <code>org.osgi.dto.DTO</code>
<code>toString</code>

Field Detail

`topic`

`public String topic`

The Event Topic

`eventData`

`public Map<String, Object> eventData`

The Data from the Event in Map form

`publicationTime`

`public Instant publicationTime`

The time at which the event was published

Constructor Detail

MonitorEvent

public MonitorEvent()

Interface TypeSafeEventMonitor

org.osgi.service.event.typed.monitor

@org.osgi.annotation.versioning.ProviderType
public interface TypeSafeEventMonitor

The EventMonitor service can be used to monitor the events that are sent using the EventBus, and that are received from remote EventBus instances

ThreadSafe

Method Summary		Page
org.osgi.util.pushstream.PushStream<MonitorEvent>	monitorEvents() Get a stream of events, starting now.	26
org.osgi.util.pushstream.PushStream<MonitorEvent>	monitorEvents(int history) Get a stream of events, including up to the requested number of historical data events.	26
org.osgi.util.pushstream.PushStream<MonitorEvent>	monitorEvents(Instant history) Get a stream of events, including historical data events prior to the supplied time	26

Method Detail

monitorEvents

[org.osgi.util.pushstream.PushStream<MonitorEvent>](#) **monitorEvents()**

Get a stream of events, starting now.

Returns:
A stream of event data

monitorEvents

[org.osgi.util.pushstream.PushStream<MonitorEvent>](#) **monitorEvents(int history)**

Get a stream of events, including up to the requested number of historical data events.

Parameters:
[history](#) - The requested number of historical events, note that fewer than this number of events may be returned if history is unavailable, or if insufficient events have been sent.

Returns:
A stream of event data

monitorEvents

[org.osgi.util.pushstream.PushStream<MonitorEvent>](#) **monitorEvents(Instant history)**

Get a stream of events, including historical data events prior to the supplied time

Parameters:

history - The requested time after which historical events, should be included. Note that events may have been discarded, or history unavailable.

Returns:

A stream of event data

Package org.osgi.service.event.typed.propertytypes

@org.osgi.annotation.versioning.Version(value="1.0.0")

Type Safe Event Component Property Types Package Version 1.0.

See:

Description

Annotation Types Summary		Page
EventTopics	Component Property Type for the EventConstants.TYPE_SAFE_EVENT_TOPICS service property of a TypedEventHandler or UntypedEventHandler service.	29
EventType	Component Property Type for the EventConstants.TYPE_SAFE_EVENT_TYPE service property of an TypedEventHandler service.	30

Package org.osgi.service.event.typed.propertytypes Description

Type Safe Event Component Property Types Package Version 1.0.

When used as annotations, component property types are processed by tools to generate Component Descriptions which are used at runtime.

Bundles wishing to use this package at runtime must list the package in the Import-Package header of the bundle's manifest.

Example import for consumers using the API in this package:

Import-Package: org.osgi.service.event.propertytypes; version="[1.0,2.0)"

Annotation Type EventTopics

org.osgi.service.event.typed.propertytypes

```
@org.osgi.service.component.annotations.ComponentPropertyType
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
@RequireTypeSafeEvents
public @interface EventTopics
```

Component Property Type for the EventConstants.TYPE_SAFE_EVENT_TOPICS service property of a TypedEventHandler OR UntypedEventHandler service.

This annotation can be used on a component to declare the values of the EventConstants.TYPE_SAFE_EVENT_TOPICS service property.

See Also:
"Component Property Types"

Required Element Summary		Page
String[]	value Service property specifying the Event topics of interest to an EventHandler service.	29

Element Detail

value

```
public abstract String[] value
```

Service property specifying the Event topics of interest to an EventHandler service.

Returns:
The event topics.

See Also:
EventConstants.TYPE_SAFE_EVENT_TOPICS

Annotation Type EventType

org.osgi.service.event.typed.propertytypes

```
@org.osgi.service.component.annotations.ComponentPropertyType
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.TYPE)
@RequireTypeSafeEvents
public @interface EventType
```

Component Property Type for the EventType.TYPE_SAFE_EVENT_TYPE service property of an TypedEventHandler service.

This annotation can be used on an TypedEventHandler component to declare the value of the EventType.TYPE_SAFE_EVENT_TYPE service property.

See Also:
"Component Property Types"

Required Element Summary		Page
String	value Service property specifying the EventType for a TypedEventHandler service.	30

Element Detail

value

```
public abstract String value
```

Service property specifying the EventType for a TypedEventHandler service.

Returns:
The event filter.

See Also:
EventType.TYPE_SAFE_EVENT_TYPE

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

Event Admin has TopicPermission, do we need an equivalent?

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