OSGi Working Group

OSGi Compendium

Release 8.1
December 2022

Licensed under the Eclipse Foundation Specification

DRAFT

Copyright © 2000, 2024 Eclipse Foundation

LICENSE

Eclipse Foundation Specification License – v1.0

By using and/or copying this document, or the Eclipse Foundation document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the Eclipse Foundation document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

- link or URL to the original Eclipse Foundation document.
- All existing copyright notices, or if one does not exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © [$date-of-document] Eclipse Foundation, Inc. <<url to this license>>"

Inclusion of the full text of this NOTICE must be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of Eclipse Foundation documents is granted pursuant to this license, except anyone may prepare and distribute derivative works and portions of this document in software that implements the specification, in supporting materials accompanying such software, and in documentation of such software, PROVIDED that all such works include the notice below. HOWEVER, the publication of derivative works of this document for use as a technical specification is expressly prohibited.

The notice is:

"Copyright © [$date-of-document] Eclipse Foundation. This software or document includes material copied from or derived from [title and URI of the Eclipse Foundation specification document]."

Disclaimers

THIS DOCUMENT IS PROVIDED "AS IS," AND THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION MAKE 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 DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of the copyright holders or the Eclipse Foundation may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

DRAFT

Table of Contents

157 Typed Event Service Specification 5

157.1 Introduction 5
157.2 Events 6
157.3 Publishing Events 8
157.4 Receiving Events 11
157.5 The Typed Event Bus Service 17
157.6 Monitoring Events 17
157.7 Capabilities 19
157.8 Security 20
157.9 org.osgi.service.typedevent 21
157.10 org.osgi.service.typedevent.annotations 27
157.11 org.osgi.service.typedevent.monitor 28
157.12 org.osgi.service.typedevent.propertytypes 31
157.13 References 33
157.14 Changes 33

Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

157 Typed Event Service Specification

Version 1.1

157.1 Introduction

Eventing systems are a common part of software programs, used to distribute information between parts of an application. To address this, the ??? was created as one of the earliest specifications defined by the OSGi Compendium. The design and usage of the Event Admin specification, however, makes certain trade-offs that do not fit well with modern application design:

- *Type Safety* - 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 to locate data, and on careful handling of data values with unknown types.
- *Unhandled Events* - Events that are sent but have no interested Event Consumers are silently discarded. There is no way to know that an event was not handled, short of disrupting the system by registering a handler for *all* events.
- *Observability* - There is no simple, non-invasive way to monitor the flow of events through the system. The ability to monitor and profile applications using Event Admin is therefore relatively limited.

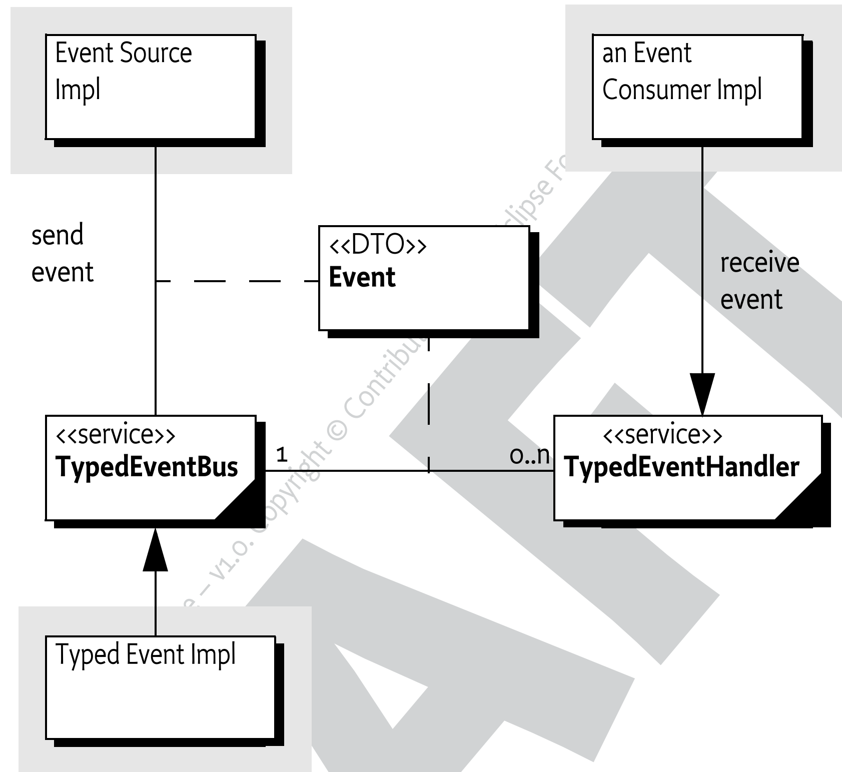
Adding these features to the original ??? specification is not feasible without breaking backward compatibility for clients. Therefore this specification exists to provide an alternative eventing model which supports these different requirements by making different design trade-offs.

157.1.1 Essentials

- *Event* - A set of data created by an Event Source, encapsulated as an object and delivered to one or more Event Consumers.
- *Event Schema* - A definition of the expected data layout within an event, including the names of data fields and the types of data that they contain.
- *Event Topic* - A String identifying the *topic* of an Event, effectively defining the Event Schema and the purpose of the event.
- *Event Source* - A software component which creates and sends events.
- *Event Consumer* - 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 Consumers.

DRAFT
Licensed under the Eclipse Foundation Specification License v1.0. Copyright © Contributors to the Eclipse Foundation.

Figure 157.1 *Class and Service overview*



157.1.2 Entities

- *Typed Event Bus* - A service registered by the Typed Event implementation that can be passed an Event object and that will distribute that event to any suitable Event Handler Services.
- *Event Handler* - A service registered by an Event Consumer suitable for receiving Event data from the Typed Event Bus.

157.2 Events

In this specification an Event is a set of string keys associated with data values. The defined set of allowable keys and permitted value types for the keys in an Event is known as the Event Schema. Both the Event Source and Event Consumers must agree on a schema, or set of compatible schemas, in order for events to be consumed correctly.

157.2.1 Type Safe Events

A Type Safe Event is one in which the Event Schema is defined as a Java class. Using a Java class provides a formal definition of the schema - event data uses field names in the class as the keys, and each field definition defines the permitted type of the value.

Type Safe Event classes are expected to either:

- Conform to OSGi DTO rules - the architecture of OSGi DTOs is described in ???. All methods, all static fields, and any non public instance fields of an event object must be ignored by the Typed Event Service when processing the Event data.

DRAFT

- Be [1] *Java Records*. Event classes that are Java Records can be identified as they are subclasses of `java.lang.Record`. The Event data consists of the record components. All other parts of a record must be ignored by the Typed Event Service when processing the Event data.

Some implementations of the Typed Event Service may support Type Safe Event classes that do not conform to these rules, transforming them as needed in an implementation specific way. This is permitted by this specification, however consumers which rely on this behaviour may not be portable between different implementations of this specification.

157.2.1.1 Nested Data Structures

OSGi DTOs are permitted to have data values which are also DTOs, allowing nested data structures to be created. This is also allowed for Type Safe Events, but with the same restriction that the event data must be a tree. There is no restriction on the depth of nesting permitted.

157.2.2 Untyped Events

An Untyped Event is one in which there is no Java class defining the Event Schema. In this case the event data is defined using a `Map` type with `String` keys and values limited to types acceptable as fields in a DTO, excepting:

- DTO types - an untyped event may not have DTOs inside it as these form part of a typed schema.
- Maps are only permitted if they follow the rules for Untyped events, that is having `String` keys and DTO restricted value types excluding DTOs.

Untyped Event instances are capable of representing exactly the same data as present in a Type Safe Event instance, and are also subject to the same restrictions, that is the data must be a tree. Nested data should be included as sub-maps within the event map, and these sub-maps may in turn contain nested data.

157.2.3 Non Standard Type Safe Events

Some Event schemas may be represented by an existing type which does not match the rules for acceptable type safe event classes. In this case there are two main options:

- Create a DTO or record representation of the event schema, and convert from the existing type into the new representation in code.
- Convert the event data into an Untyped Event representation using nested Maps.

For example, the following code demonstrates how an object following the JavaBeans pattern can be converted into a DTO type or an untyped map:

```
public class ExampleJavaBean {
    private String message;

    public String getMessage() { return message; }

    public void setMessage(String message) { this.message = message; }
}

public class ExampleEvent {
    public String message;
}

@Component
public class ExampleEventSource {
    private ExampleEvent createEventFromJavaBean(ExampleJavaBean bean) {
```

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0 – Copyright © Contributors to the Eclipse Foundation

```
        return Converters.standardConverter().convert(bean)
                .to(ExampleEvent.class);
    }

    private Map<String, Object> createMapFromJavaBean(ExampleJavaBean bean) {
        return Converters.standardConverter().convert(bean)
                .to(new TypeReference<Map<String, Object>>(){});
    }
}
```

157.2.4 Event Mutability and Thread Safety

The Typed Event Service is inherently multi-threaded. Events may be published from multiple threads, and event data may be delivered to consumers on multiple threads. Event Sources and Event Consumers must therefore assume that event data is shared between threads from the moment that it is first passed to the TypedEventBus.

157.2.4.1 Typed Event Mutability

Typed Events, and in particular DTO types, provide a simple yet powerful mechanism for defining an Event Schema in a type-safe way. However their use of mutable public fields means that they are potentially dangerous when shared between threads. Event Sources and Event Consumers should assume that their event instances are shared between threads and therefore not mutate the event data after publication or receipt.

If an Event Handler does need to make changes to an incoming event then it must copy the event data into a new DTO instance. Note that any nested DTO values in the event data must also be copied if they are to be mutated.

157.2.4.2 Untyped Event Mutability

When an event source publishes untyped event data, it passes a Map instance to the Typed Event Bus. The Typed Event Bus is not required to take a copy of this Map, and therefore the event source must not change the Map, or any data structures within the Map, after the call to deliverUntyped(String,Map).

Untyped Events are delivered as implementations of the Map interface. Bundles consuming untyped events should not rely on the event object being any particular implementation of Map, and should treat the event object as immutable. The Typed Event Bus implementation may make copies of the event data, or enforce the immutability of the map, before passing the event data to an Event Handler.

157.3 Publishing Events

To publish an event, the Event Source must retrieve the Typed Event Bus service from the OSGi service registry. The Event Source then creates an event object and calls one of the Typed Event Bus service's methods to publish the event. Event publication is asynchronous, meaning that when a call to the Typed Event Bus returns there is no guarantee that all, or even any, listeners have been notified.

157.3.1 Event Topics

Events are always published to a topic. The topic of an event defines the *schema* of the event. Topics exist in order to give Event Consumers information about the schema of the event, and the opportunity to register for just the events they are interested in. When a topic is designed, its name should not include any other information, such as the publisher of the event or the data associated with the event, those parts are intended to be stored in the event properties.

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0, Copyright © Contributors to the Eclipse Foundation

The topic therefore serves as a first-level filter for determining which handlers should receive the event. Typed Event service implementations use the structure of the topic to optimize the dispatching of the events to the handlers. The following example code demonstrates how to send an event to a topic.

```
public class ExampleEvent {
    public String message;
}

@Component
public class ExampleEventSource {
    @Reference
    TypedEventBus bus;

    public void sendEvent() {
        ExampleEvent event = new ExampleEvent();
        event.message = "The time is " + LocalDateTime.now();

        bus.deliver("org/osgi/example/ExampleEvent", event);
    }
}
```

Topics are arranged in a hierarchical namespace. Each level is defined by a token and levels are separated by solidi ('/' \u002F). More precisely, the topic must conform to the following grammar:

```
// For further information see General Syntax Definitions in Core

topictoken ::: ( jletterordigit | '-' ) +

topic      ::= topictoken ( '/' topictoken ) *
```

Topics should be designed to become more specific when going from left to right. Consumers can provide a prefix that matches a topic, using the preferred order allows a handler to minimize the number of prefixes it needs to register.

Topics are case-sensitive. As a convention, topics should follow the reverse domain name scheme used by Java packages to guarantee uniqueness. The separator must be a solidus ('/' \u002F) instead of the full stop ('.' \u002E).

This specification uses the convention fully/qualified/package/ClassName/ACTION. If necessary, a pseudo-class-name is used.

Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT

157.3.2 Automatically Generated Topics

In many cases the name of a topic contains no information other than defining the schema of the events sent on that topic. Therefore, when publishing a Typed Event to the Typed Event Bus, the Typed Event implementation is able to automatically generate a topic name based on the the type of the event object being published.

For the deliver(Object) method on the Typed Event Bus where no topic string is provided, the implementation must create a topic string using the fully qualified class name of the event object. To convert the class name into a valid topic the full stop . separators must be converted into solidus / separators. A non-normative example implementation follows:

```
public void deliver(Object event) {
    String topicName = event.getClass().getName().replace('.', '/');

    this.deliver(topicName, event);
```

```
}
```

The following example demonstrates how an Event Source can make use of an automatically generated topic name.

```
package org.osgi.example;

public class ExampleEvent {
    public String message;
}

@Component
public class ExampleEventSource {
    @Reference
    TypedEventBus bus;

    public void sendEvent() {
        ExampleEvent event = new ExampleEvent();
        event.message = "The time is " + LocalDateTime.now();

        // This event will be delivered to the
        // topic "org/osgi/example/ExampleEvent"
        bus.deliver(event);
    }
}
```

157.3.3 Thread Safety

The TypedEventBus implementation must be thread safe and allow for simultaneous event publication from multiple threads. For any given source thread, events must be delivered in the same order as they were published by that thread. Events published by different threads, however, may be delivered in a different order from the one in which they were published.

For example, if thread *A* publishes events *1*, *2* and *3*, while thread *B* publishes events *4*, *5* and *6*, then the events may be delivered:

- *1*, *2*, *3*, *4*, *5*, *6*
- *4*, *1*, *2*, *5*, *6*, *3*
- and so on

but events will never be delivered *1*, *2*, *6*, *4*, *5*, *3*

157.3.4 Typed Event Publishers

Typical Event Sources publish single events at irregular intervals. These event sources are best served by using the deliver methods on the TypedEventBus. Some Event Sources, however, produce bursts of events, or frequent, regular events to a topic. In these cases there can be overhead associated with validating the topic and security context each time an event is published.

A TypedEventPublisher is associated with a single topic which is set during creation. All events published using the TypedEventPublisher are delivered to that topic. A TypedEventPublisher can be obtained from the TypedEventBus using one of the createPublisher methods.

A TypedEventPublisher is AutoCloseable and the isOpen method may be used to determine whether it has been closed. Once closed the object may no longer be used to publish events, and all event delivery methods will throw IllegalStateException.

```
public class ExampleEvent {
    public String message;
```

DRAFT
Licensed under the Eclipse Foundation Specification License – Copyright © Contributors to the Eclipse Foundation.

```
}

@Component
public class ExampleEventSource {
    @Reference
    TypedEventBus bus;

    public void sendEvents() {

        try(TypedEventPublisher<ExampleEvent> publisher
                = bus.createPublisher("my/topic/name", ExampleEvent.class)) {

            for(int i = 0; i < 100; i++) {
                ExampleEvent event = new ExampleEvent();
                event.message = "The event is " + i;
                publisher.deliver(event);
            }
        }
    }
}
```

Because the topic for a TypedEventPublisher is defined at creation time the implementation may only validate it once. Also, the TypedEventBus implementation must check the caller's permission to publish to the topic during the call to createPublisher, but should not check it again for each delivery. Note that this means an Event Source *must not* share a TypedEventPublisher outside its own bundle.

157.4 Receiving Events

Event Consumers can receive events by registering an appropriate Event Handler service in the Service Registry. This is a TypedEventHandler to receive events as type-safe objects, or an UntypedEventHandler to receive events as untyped Map structures.

Published events are then delivered, using the whiteboard pattern, to any Event Handler service which has registered interest in the topic to which the event was published.

157.4.1 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 Typed Event implementation must adapt the Event object into the type defined by the TypedEventHandler implementation. Implementations of this specification are free to choose their own adaptation mechanism, however it must guarantee at least the same functionality as ???.

A simple example of receiving a typed event follows:

```
public class ExampleEvent {
    public String message;
}

@Component
```

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

```
public class ExampleTypedConsumer implements TypedEventHandler<ExampleEvent> {
    @Override
    public void notify(String topic, ExampleEvent event) {
        System.out.println("Received event: " + event.message);
    }
}
```

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 service property. This property has a string value containing the fully-qualified type name of the type that the Typed Event Handler expects to receive. This type must be loaded by the Typed Event implementation using the classloader of the bundle which registered the Typed Event Handler service. The loaded type must then be used as the target type when converting events. For example:

```
public class ExampleEvent {
    public String message;
}

public class SpecialisedExampleEvent extends ExampleEvent {
    public int sequenceId = Integer.MIN_VALUE;
}

@Component
@EventType(SpecialisedExampleEvent.class)
public class ExampleTypedConsumer implements TypedEventHandler<ExampleEvent> {
    @Override
    public void notify(String topic, ExampleEvent event) {
        System.out.println("Received event: " + event.message);

        // The event will always be of type SpecialisedExampleEvent
        System.out.println("Event sequence id was " +
            ((SpecialisedExampleEvent) event).sequenceId);
    }
}
```

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 topic instead of the reified type. To use a specific named topic the Typed Event Handler service may be registered with an event.topics service property specifying the topic(s) as a String+ value.

```
public class ExampleEvent {
    public String message;
}

@Component
@EventTopics({"foo", "foo/bar"})
public class ExampleTypedConsumer implements TypedEventHandler<ExampleEvent> {
    @Override
    public void notify(String topic, ExampleEvent event) {
        System.out.println("Event received on topic: " + topic +
                " with message: " + event.message);
    }
}
```

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

157.4.2 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 Untyped Event Handler implementation must be registered as a service in the service registry using the UntypedEventHandler interface.

When delivering an event to an Untyped Event Handler the Typed Event Service must, if necessary, convert the event data to a nested map structure.

The event.topics service property must be used when registering an Untyped Event Hander service. If it is not, then no events will be delivered to that Untyped Event Handler service.

```
public class ExampleEvent {
    public String message;
}

@Component
@EventTopics({"foo", "foo/bar"})
public class ExampleUntypedConsumer implements UntypedEventHandler {
    @Override
    public void notifyUntyped(String topic, Map<String,Object> event) {
        System.out.println("Event received on topic: " + topic
                + " with message: " + event.get("message"));
    }
}
```

157.4.3 Wildcard Topics

The event.topics property may contain one or more wildcard topics. These are Strings which contain a topic name including one or more wildcards.

157.4.3.1 Single-Level Wildcards

A Single-Level wildcard is used to match all topics which differ only by a single topic token within the topic String. This is done by using a "+" character to indicate the topic token which may vary.

Using a single-level wildcard means that the Event Handler must be called Events sent to topics matching the rest of the topic String. For example the component:

```
@Component
@EventTopics("foo/+/foobar")
public class ExampleUntypedConsumer implements UntypedEventHandler {
    @Override
    public void notifyUntyped(String topic, Map<String,Object> event) {
        System.out.println("Event received on topic: " + topic
                + " with message: " + event.get("message"));
    }
}
```

would receive events sent to the topics foo/bar/foobar and foo/baz/foobar, but not the topics foo/foobar or foo/bar/foobar/baz.

The + character in a wildcard topic must always be the only character in the topic token that it matches, meaning that topic names such as foo+/bar and foo/+bar are not valid. It is valid to use the topic name + to receive events on *all* single level topics.

DRAFT Licensed under the Eclipse Foundation Specification License – with Copyright © Contributors to the Eclipse Foundation

157.4.3.2 The Multi-Level Wildcard

The multi-level wildcard is used to match all topics which share a given prefix. This is done by appending "/*". to the prefix. This value means that the Event Handler must be called for Events sent to sub-topics of the named topic. For example the component:

```
@Component
@EventTopics("foo/*")
public class ExampleUntypedConsumer implements UntypedEventHandler {
    @Override
    public void notifyUntyped(String topic, Map<String,Object> event) {
        System.out.println("Event received on topic: " + topic
                + " with message: " + event.get("message"));
    }
}
```

would receive events sent to the topics foo/bar and foo/baz, but not the topics foo or foobar/fizzbuzz.

The * character in a wildcard topic must always follow a solidus / character, and must be the final character in the topic string, meaning that topic names such as foo* and foo/*/bar are not valid. The only exception to this rule is that it is valid to use the topic name * to receive events on *all* topics. While it is valid to do so, using the topic * is not typically recommended. For a mechanism to monitor the events flowing through the system see *Monitoring Events* on page 17.

157.4.4 Unhandled Events

Unhandled Events are events sent by an Event Source but which have no Event Handler service listening to their topic. Rather than these events being discarded, the Typed Event implementation will search the service registry for services implementing UnhandledEventHandler.

If any services are found then the Typed Event implementation will call the notifyUnhandled method passing the topic name and event data to all of the registered Unhandled Event Handler services.

```
public class ExampleEvent {
    public String message;
}

@Component
public class ExampleUnhandledConsumer implements UnhandledEventHandler {
    @Override
    public void notifyUnhandled(String topic, Map<String,Object> event) {
        System.out.println("Unhandled Event received on topic: " + topic);
    }
}
```

157.4.5 Filtering Events

Sometimes the use of a topic is insufficient to restrict the events received by an event consumer. In these cases the consumer can further restrict the events that they receive by using a filter. The filter is supplied using the event.filter service property, the value of which is an LDAP filter string. This filter is applied to the event data, and only events which match the filter are delivered to the event handler service.

157.4.5.1 Nested Event Data

Complex events may contain nested data structures, such as DTOs, as values in the event data. As LDAP filtering is only designed to match against simple data this means that some event properties

Licensed under the Eclipse Foundation Specification License – 1.0. Copyright © Contributors to the Eclipse Foundation.
DRAFT

cannot be filtered using the event.filter property. The event filter is therefore only suitable for use in matching top-level event properties.

157.4.5.2 Ignored Events

Note that the use of a filter is different from receiving an event and choosing to ignore it based on its data. If an event fails to match the filter supplied by an event handler service then it is *not delivered* to that event handler. This means that the event data remains eligible to be sent to an UnhandledEventHandler unless another event handler does receive it. An event that is received, but ignored, by an event handler service *does* count as having been delivered, and so will never be sent to an UnhandledEventHandler.

157.4.6 Failing Event Handlers

Event Handler implementations are called by the Typed Event Bus implementation, and are expected:

- Not to throw exceptions from their callback method
- To return quickly - any long running tasks should be moved to another thread

If a Typed Event Bus implementation detects an Event Handler that is behaving incorrectly, either by throwing exceptions, or by taking a long time to process the event, or some other problem, then the implementation may block further event delivery to that Event Handler.

If an Event Handler is blocked by the event implementation then this situation must be logged. Also, if a blocked Event Handler service is updated then the block must be removed by the implementation. If the updated service continues to behave incorrectly then the block may be reinstated.

157.4.7 Event History

Event Handlers may be registered at any time, which can lead to ordering problems with respect to Event Sources publishing their first events. Specifically a late arriving Event Handler may miss the first message(s) from an early registering Event Source.

While this situation can be detected by the Event Source (for example by using an Unhandled Event Handler) or by the Event Handler (for example by using the TypedEventMonitor) doing so is relatively verbose.

In simple situations, such as watching the value of an irregularly updating Event Source, it is much easier for the Event Handler to have the last event(s) replayed to it before it starts receiving newly published events. This behaviour can be enabled using the TYPED_EVENT_HISTORY service property of the Event Handler.

The event.history property is of type Integer and must be greater than or equal to zero. The value of the property defines the maximum number of historical events that should be replayed to the Event Handler before normal event delivery begins. If the event.history property is not set then it should be assumed to have the value zero.

When the Typed Safe Events implementation detects the registration of an Event Handler service which has requested that one or more historical events should be replayed then it should follow the following non normative steps:

1. Register the Event Handler service, but delay any event delivery until historical event delivery has completed.
2. Identify events in the retained history which match the event.topics defined by the Event Handler.
3. Filter the identified events using the event.filter defined by the Event Handler.
4. Sort the remaining events into chronological order.
5. Replay the last N events, in chronological order, where N is the number of events requested by the Event Handler.

Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.
DRAFT

6. Enable the Event Handler service and begin delivering any delayed events.

The above algorithm is intended to be illustrative rather than normative and it is expected that implementations will optimise their approaches.

As retaining history is an optional feature of the Type Safe Events implementation, and even where history is retained there may not be any history to replay, it is possible that there will be insufficient events to supply the number requested in event.history. This situation is not an error, and an Event Handler service must not require that any historical events are delivered to it. The only guarantee is that no more than event.history historical events will be delivered before normal event delivery commences.

157.4.8 Event Handler Service Properties

The service properties that can be used to configure an Event Handler service are outlined in the following table.

Table 157.1 Service properties applicable to Event Handler services

Service Property Name	Type	Description
event.topics	String+	Declares the topic pattern(s) for which the service should be called. This service property is *required* for UntypedEventHandler services, but TypedEventHandler services may omit it if they are only interested in the default topic name for their reified type. UnhandledEventHandler services may use this property to restrict the events that they receive. See TYPED_EVENT_TOPICS.
event.type	String	Defines the target type into which events should be converted before being passed to the Event Handler service. This service property is *forbidden* for UntypedEventHandler and UnhandledEventHandler services, but TypedEventHandler services may use it if they wish to further refine the type of data they wish to receive. See TYPED_EVENT_TYPE.
event.filter	String	Defines an LDAP filter which should be tested against the properties in the event data. Only events which pass the filter will be passed to the the Event Handler service. This service property is *optional* and permitted for TypedEventHandler, UntypedEventHandler and UnhandledEventHandler services. See TYPED_EVENT_FILTER.
event.history	Integer	Defines the number of historical events which should be replayed to the Event Handler Service before normal event delivery begins. The value of this service property must be greater than or equal to zero. This service property is *optional* and permitted for both TypedEventHandler and UntypedEventHandler services but *not* for UnhandledEventHandler services. See TYPED_EVENT_HISTORY.

157.4.9 Error Handling

There are several possible error scenarios for Event Handlers:

- TypedEventHandler - If the target event type is not discoverable, that is there is no reified type information, nor is there an event.type property, then the target type for the event is not known. In this situation there is no way for the Typed Event implementation to correctly target an event schema, and the TypedEventHandler must be ignored. The implementation must write a message to the log indicating which service is being ignored.

DRAFT
Licensed under the Eclipse Foundation Specification License v1.0 Copyright © Contributors to the Eclipse Foundation.

- TypedEventHandler - If the target event type is discoverable but cannot be loaded using the class loader of the bundle which registered the Typed Event Handler service then there is no way for the Typed Event implementation to correctly target an event schema, and the Event Handler must be ignored. The implementation must write a message to the log indicating which service is being ignored.
- All Handler Types - If the event data cannot be adapted to the target type, that is the incoming data cannot be transformed due to badly mismatched property names or values, then that specific Event cannot be submitted to the Handler. The Typed Event implementation must write a message to the log indicating which service failed to receive the event. If this error occurs repeatedly then the Typed Event implementation may choose to deny list and ignore the Event Handler service. Deny listing decisions must be written to the log.
- All Handler Types - If the event.topics property contains one or more invalid values then the Event Handler service must be ignored. The implementation must write a message to the log indicating which service is being ignored.
- All Handler Types - If the event.filter property contains an invalid value then the Event Handler service must be ignored. The implementation must write a message to the log indicating which service is being ignored.
- All Handler Types - If the event.history property contains an invalid value then the Event Handler service must be ignored. The implementation must write a message to the log indicating which service is being ignored.

157.5 The Typed Event Bus Service

The Typed Event implementation must register a Typed Event Bus service in the service registry. This service must implement and advertise the TypedEventBus interface.

157.5.1 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 there are problems delivering the event. The Event Bus service should not throw exceptions from any publication methods except:

- NullPointerException if the event data is null.
- IllegalArgumentException if a topic name is supplied and it violates the topic name syntax.

157.6 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 Typed Event implementation must register a TypedEventMonitor service which can be used to monitor the flow of events through the Event Bus.

Events flowing through the Typed Event Bus can be monitored using one of the monitorEvents methods from the TypedEventMonitor service. These methods return a ??? which delivers MonitorEvent instances each time an event is sent via the TypedEventBus. The monitor events contain the event topic, the event data, and a timestamp indicating when the event was sent.

157.6.1 Event History

In a running system it is often useful for monitoring tools to replay recent data immediately after a problem has occurred. For that reason Typed Event Monitor instances may store past events so that

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

they can be replayed if requested. There are four monitorEvents methods capable of replaying history:

- monitorEvents(int) takes an int representing the number of past events that should be replayed from the cached history. This stream remains connected after replaying history and will continue to receive new events.
- monitorEvents(int,boolean) takes an int representing the number of past events that should be replayed from the cached history and a boolean indicating whether to remain connected after all available history has been replayed. Passing true will ensure that only historical events are replayed.
- monitorEvents(Instant) takes an Instant, representing the time in the past from which the stream of monitoring events should start.
- monitorEvents(Instant,boolean) takes an Instant, representing the time in the past from which the stream of monitoring events should start and a boolean indicating whether to remain connected after all available history has been replayed. Passing true will ensure that only historical events are replayed.

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.

157.6.2 Configuring Event History

It is not usually the case that the history for every event topic is equally important. In fact for many topics history is unimportant, while for others it is highly desirable to keep a small number of historical messages indefinitely. To support this the Typed Event Service implementation allows for history storage to be configured on a per-topic basis, with both a minimum and maximum event requirement. The minimum requirement guarantees that at least N events are kept for the topic and not discarded in favour of events on other topics. The maximum requirement provides a ceiling on the number of events that will be kept in history for a given topic, ensuring that one busy topic does not dominate the whole history.

As the primary means of accessing historical events the Typed Event Monitor service also provides the ability to introspect and customise the historical event storage.

157.6.2.1 Introspecting Event History Configuration

The most basic feature of the Event History Configuration is the total event storage capacity. This is typically set in configuration during startup and cannot necessarily easily be changed at runtime. The total event storage capacity can be queried by calling getMaximumEventStorage().

The global history configuration is available from the getConfiguredHistoryStorage() method. This returns the entire history retention policy for the Typed Event Service, individual configuration entries can be obtained by calling getConfiguredHistoryStorage(String). The returned configuration provides an Entry where the key is the minimum number of events to keep and the value is the maximum.

Configuration entries use topic filters rather than topic names, and are therefore permitted to use wildcards. This can therefore mean that more than one configuration matches a given topic name. To ensure consistency between implementations the following disambiguation algorithm applies:

1. Identify the set of matching topic filters from the configuration
2. Extract the first token from each of the filters
3. Retain the most preferred filters based on the following preference order:
   a. An exact token match
   b. A single level wildcard token match
   c. A multi-level wildcard match

DRAFT
Licensed under the Eclipse Foundation Specification License v1.0. Copyright © Contributors to the Eclipse Foundation

4. If only one filter remains then this defines the configuration to use, otherwise remove the leading token from the retained topic filters and repeat this algorithm from the second step.

When obtaining the global configuration it is returned as a Map with the configured topic filters used as keys. Furthermore the keys in the map are *ordered* such that the most specific match for a given topic will *always* be encountered first. This can be used in conjunction with the topicFilterMatches(String) or topicFilterMatches(String,String) to easily identify which policy would apply to a given topic.

In addition to determining the global history configuration it is also possible to introspect the specific configuration for event history retention. For example, to find the policy for a named topic the getEffectiveHistoryStorage(String) method will, as for the global configuration, return an Entry where the key is the minimum number of events to keep and the value is the maximum. This method must take into account any wildcards and precedence in the global configuration to determine the correct policy for the supplied topic name. Note that this method can only be used for topic names, and passing a topic filter containing wildcards will raise an IllegalArgumentException.

157.6.2.2 Modifying Event History Configuration

In addition to viewing the currently configured history storage configuration users are able to programmatically modify the configuration at runtime.

Adding or changing the configuration for a given topic filter is possible using the configureHistoryStorage(String,int,int) method. This is used to set the required minimum and maximum numbers of events that should be stored for each topic that matches the supplied filter. It is invalid to pass a number less than zero for either the minimum or maximum number of events, and it is also invalid to set a minimum greater than the maximum. In both cases an IllegalArgumentException must be raised by the implementation.

When a configuration is added or updated the implementation must ensure that sufficient space exists to be able to hold at least the minimum number of events for each configured topic. If the addition or update would breach the getMaximumEventStorage() then an IllegalStateException must be thrown by the implementation. For this reason it is also invalid to set a non zero minimum number of events when using a wildcard topic filter, as doing so makes it impossible for the implementation to know how many topics the minimum will apply to. Passing a non-zero minimum with a wildcard topic filter must therefore also trigger an IllegalArgumentException. Assuming that sufficient space is available the return value of the configureHistoryStorage method indicates the storage space available for this topic filter. This must be at least the minimum requirement, and at most the maximum requirement, and is calculated by the implementation taking into account the total available storage, and the pre-existing configuration.

If no longer required then the configuration for a given topic filter can be deleted using removeHistoryStorage(String). When this method is called it may make zero or more historical events eligible for removal from the history, depending on the other configured filters. The implementation should ensure that any superfluous events are promptly removed from the event history and must not return them in any subsequent requests for historical data.

157.7 Capabilities

157.7.1 osgi.implementation Capability

The Typed Event implementation bundle must provide the osgi.implementation capability with the name TYPED_EVENT_IMPLEMENTATION. This capability can be used by provisioning tools and during resolution to ensure that a Typed Event implementation is present. The capability must also declare a uses constraint for the org.osgi.service.typedevent package and provide the version of this specification:

DRAFT Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

```
Provide-Capability: osgi.implementation;
 osgi.implementation="osgi.typedevent";
 uses:="org.osgi.service.typedevent";
 version:Version="1.1"
```

The RequireTypedEvent annotation can be used to require this capability.

This capability must follow the rules defined for the ???.

157.7.2 osgi.service Capability

The bundle providing the Typed Event Bus service must provide capabilities in the osgi.service namespace representing the services it is required to register. This capability must also declare uses constraints for the relevant service packages:

```
Provide-Capability: osgi.service;
 objectClass:List<String>="org.osgi.service.typedevent.TypedEventBus";
 uses:="org.osgi.service.typedevent",
 osgi.service;
 objectClass:List<String>="org.osgi.service.typedevent.monitor.TypedEventMonitor";
 uses:="org.osgi.service.typedevent.monitor"
```

This capability must follow the rules defined for the ???.

157.8 Security

157.8.1 Topic Permission

The TopicPermission class allows fine-grained control over which bundles may post events to a given topic and which bundles may receive those events.

The target parameter for the permission is the topic name. TopicPermission classes uses a wildcard matching algorithm similar to the BasicPermission class, except that solidi ('/' \u002F) are used as separators instead of full stop characters. For example, a name of a/b/* implies a/b/c but not x/y/z or a/b.

There are two available actions: PUBLISH and SUBSCRIBE. These control a bundle's ability to either publish or receive events, respectively. Neither one implies the other.

157.8.2 Required Permissions

Bundles that need to consume events must be granted permission to register the appropriate handler service. For Example: ServicePermission[org.osgi.service.typedevent.TypedEventHandler, REGISTER] or ServicePermission[org.osgi.service.typedevent.UntypedEventHandler, REGISTER] or ServicePermission[org.osgi.service.typedevent.UnhandledEventHandler, REGISTER]. In addition, bundles that consume events require TopicPermission[<topic>, SUBSCRIBE] for each topic they want to be notified about.

Bundles that need to publish events must be granted permission to get the TypedEventBus service, that is ServicePermission[org.osgi.service.typedevent.TypedEventBus, GET] so that they may retrieve the Typed Event Bus and use it. In addition, event sources require TopicPermission[<topic>, PUBLISH] for each topic they want to send events to. This includes any default topic names that are used when publishing

Bundles that need to monitor events flowing through the bus must be granted permission to get the TypedEventMonitor service, that is ServicePermission[org.osgi.service.typedevent.monitor.TypedEventMonitor, GET] so that they may retrieve the Typed Event Monitor and use it.

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

Only a bundle that provides a Typed Event implementation should be granted ServicePermission[org.osgi.service.typedevent.TypedEventBus, REGISTER] and ServicePermission[org.osgi.service.typedevent.monitor.TypedEventMonitor, REGISTER] to register the services defined by this specification.

The Typed Event implementation must be granted ServicePermission[org.osgi.service.typedevent.TypedEventHandler, GET], ServicePermission[org.osgi.service.typedevent.UntypedEventHandler, GET], ServicePermission[org.osgi.service.typedevent.UnhandledEventHandler, GET], ServicePermission[org.osgi.service.typedevent.TypedEventBus, REGISTER] and ServicePermission[org.osgi.service.typedevent.monitor.TypedEventMonitor, REGISTER] as these actions are all required to implement the specification.

157.8.3 Security Context During Event Callbacks

During an event notification, the Typed Event implementation's Protection Domain will be on the stack above the handler's Protection Domain. Therefore, if a handler needs to perform a secure operation using its own privileges, it must invoke the doPrivileged method to isolate its security context from that of its caller.

The event delivery mechanism must not wrap event notifications in a doPrivileged call.

157.9 org.osgi.service.typedevent

Typed 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.typedevent; version="[1.0,2.0)"

Example import for providers implementing the API in this package:

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

157.9.1 Summary

- TopicPermission - A bundle's authority to publish or subscribe to typed events on a topic.
- TypedEventBus - The Typed Event service.
- TypedEventConstants - Defines standard names for Typed Event properties.
- TypedEventHandler - Listener for Typed Events.
- TypedEventPublisher - A Typed Event publisher for a single topic.
- UnhandledEventHandler - Listener for Unhandled Events.
- UntypedEventHandler - Listener for Untyped Events.

157.9.2 public final class TopicPermission extends Permission

A bundle's authority to publish or subscribe to typed events on a topic.

A topic is a slash-separated string that defines a topic.

For example:

```
org / osgi / service / foo / FooEvent / ACTION
```

DRAFT
Licensed under the Eclipse Foundation Specification License v1.0. Copyright © Contributors to the Eclipse Foundation.

Topics may also be given a default name based on the event type that is published to the topic. These use the fully qualified class name of the event object as the name of the topic.

For example:

```
com.acme.foo.event.EventData
```

TopicPermission has two actions: publish and subscribe.

Concurrency Thread-safe

157.9.2.1 public static final String PUBLISH = "publish"

The action string publish.

157.9.2.2 public static final String SUBSCRIBE = "subscribe"

The action string subscribe.

157.9.2.3 public TopicPermission(String name, String actions)

name Topic name.

actions publish,subscribe (canonical order).

- Defines the authority to publish and/or subscribe to a topic within the Typed Event service specification.

The name is specified as a slash-separated string. Wildcards may be used. For example:

```
org/osgi/service/fooFooEvent/ACTION
com/isv/*
*
```

A bundle that needs to publish events on a topic must have the appropriate TopicPermission for that topic; similarly, a bundle that needs to subscribe to events on a topic must have the appropriate TopicPermssion for that topic.

157.9.2.4 public boolean equals(Object obj)

obj The object to test for equality with this TopicPermission object.

- Determines the equality of two TopicPermission objects. This method checks that specified TopicPermission has the same topic name and actions as this TopicPermission object.

Returns true if obj is a TopicPermission, and has the same topic name and actions as this TopicPermission object; false otherwise.

157.9.2.5 public String getActions()

- Returns the canonical string representation of the TopicPermission actions.

Always returns present TopicPermission actions in the following order: publish,subscribe.

Returns Canonical string representation of the TopicPermission actions.

157.9.2.6 public int hashCode()

- Returns the hash code value for this object.

Returns A hash code value for this object.

157.9.2.7 public boolean implies(Permission p)

p The target permission to interrogate.

DRAFT Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

□ Determines if the specified permission is implied by this object.

This method checks that the topic name of the target is implied by the topic name of this object. The list of TopicPermission actions must either match or allow for the list of the target object to imply the target TopicPermission action.

```
x/y/*,"publish" -> x/y/z,"publish" is true
*,"subscribe" -> x/y,"subscribe"   is true
*,"publish" -> x/y,"subscribe"     is false
x/y,"publish" -> x/y/z,"publish"   is false
```

Returns true if the specified TopicPermission action is implied by this object; false otherwise.

157.9.2.8 public PermissionCollection newPermissionCollection()

□ Returns a new PermissionCollection object suitable for storing TopicPermission objects.

Returns A new PermissionCollection object.

157.9.3 public interface TypedEventBus

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

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

157.9.3.1 public TypedEventPublisher<T> createPublisher(Class<T> eventType)

Type Parameters <T>

<T> The type of events to be sent by the TypedEventPublisher

eventType The type of events to be sent by the TypedEventPublisher

□ Creates a TypedEventPublisher for the topic automatically generated from the passed in event type.

Returns A TypedEventPublisher that will publish events to the topic name automatically generated from eventType

Since 1.1

157.9.3.2 public TypedEventPublisher<T> createPublisher(String topic, Class<T> eventType)

Type Parameters <T>

<T> The type of events to be sent by the TypedEventPublisher

topic The topic to publish events to

eventType The type of events to be sent by the TypedEventPublisher

□ Creates a TypedEventPublisher for the supplied topic and event type

Returns A TypedEventPublisher that will publish events to the supplied topic

Since 1.1

157.9.3.3 public TypedEventPublisher<Object> createPublisher(String topic)

topic The topic to publish events to

□ Creates a TypedEventPublisher for the supplied topic

Returns A TypedEventPublisher that will publish events to the supplied topic

Since 1.1

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

157.9.3.4 public void deliver(Object event)

event The event to send to all listeners which subscribe to the topic of the 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().replace('.', '/'), event)

Throws NullPointerException– if the event object is null

157.9.3.5 public void deliver(String topic, Object event)

topic The topic to which this event should be sent.

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

- 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.

Throws NullPointerException– if the event object is null

IllegalArgumentException– if the topic name is not valid

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

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.

- 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.

Throws NullPointerException– if the event map is null

IllegalArgumentException– if the topic name is not valid

157.9.4 public final class TypedEventConstants

Defines standard names for Typed Event properties.

Provider Type Consumers of this API must not implement this type

157.9.4.1 public static final String TYPED_EVENT_FILTER = "event.filter"

The name of the service property used to indicate a filter that should be applied to events from the TYPED_EVENT_TOPICS. Only events which match the filter will be delivered to the Event Handler service.

If this service property is not present then all events from the topic(s) will be delivered to the Event Handler service.

157.9.4.2 public static final String TYPED_EVENT_HISTORY = "event.history"

The name of the service property used to indicate that an event handler would like to receive one or more historical events matching its TYPED_EVENT_TOPICS and TYPED_EVENT_FILTER before receiving any new data. The value of this property is an Integer greater than or equal to zero.

If this property is set then when the event hander is discovered by the whiteboard the event handler will deliver, in order, up to the requested number of historical events. This will occur before the delivery of any new events. If no history is available then zero events will be delivered. If insufficient history is available then fewer events than requested will be delivered.

Licensed under the Eclipse Foundation Specification License – v1.0 Copyright © Contributors to the Eclipse Foundation DRAFT

157.9.4.3 public static final String TYPED_EVENT_IMPLEMENTATION = "osgi.typedevent"

The name of the implementation capability for the Typed Event specification

157.9.4.4 public static final String TYPED_EVENT_SPECIFICATION_VERSION = "1.1"

The version of the implementation capability for the Typed Event specification

157.9.4.5 public static final String TYPED_EVENT_TOPICS = "event.topics"

The name of the service property used to indicate the topic(s) to which an a TypedEventHandler, UntypedEventHandler or UnhandledEventHandler service is listening.

If this service property is not present then the reified type parameter from the TypedEventHandler implementation class will be used to determine the topic.

157.9.4.6 public static final String TYPED_EVENT_TYPE = "event.type"

The name of the service property used to indicate the type of the event objects received by a TypedEventHandler service.

If this service property is not present then the reified type parameter from the TypedEventHandler implementation class will be used.

157.9.5 public interface TypedEventHandler<T>

<T> The type of the event to be received

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 TypedEventConstants.TYPED_EVENT_TOPICS service property.

`TypedEventHandler` objects may be registered with a service property TypedEventConstants.TYPED_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);
```

Concurrency Thread-safe

157.9.5.1 public void notify(String topic, T event)

topic The topic to which the event was sent

event The event that occurred.

□ Called by the TypedEventBus service to notify the listener of an event.

157.9.6 public interface TypedEventPublisher<T> extends AutoCloseable

<T> The type of events that may be sent using this publisher

Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.
DRAFT

A Typed Event publisher for a single topic. Bundles wishing to publish events may obtain this object from the TypedEventBus.

All events published by this publisher are sent to the same topic, as returned by getTopic().

This object should not be shared as it will not re-validate the caller's permission to send to the target topic.

Since 1.1

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

157.9.6.1 public void close()

□ Closes this TypedEventPublisher so that no further events can be sent. After closure all delivery methods throw IllegalStateException.

Calling close() on a closed TypedEventPublisher has no effect.

157.9.6.2 public void deliver(T event)

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

Throws NullPointerException– if the event object is null

IllegalStateException– if the TypedEventPublisher has been closed

157.9.6.3 public void deliverUntyped(Map<String, ?> event)

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

□ 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.

Throws NullPointerException– if the event map is null

IllegalStateException– if the TypedEventPublisher has been closed

157.9.6.4 public String getTopic()

□ Get the topic for this TypedEventPublisher. This topic is set when the TypedEventPublisher is created and cannot be changed.

Returns The topic for this TypedEventPublisher

157.9.6.5 public boolean isOpen()

□ This method allows the caller to check whether the TypedEventPublisher has been closed.

Returns false if the TypedEventPublisher has been closed, true otherwise.

157.9.7 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

Concurrency Thread-safe

DRAFT Licensed under the Eclipse Foundation Specification License v1.0. Copyright © Contributors to the Eclipse Foundation

157.9.7.1 public void notifyUnhandled(String topic, Map<String, Object> event)

topic The topic to which the event was sent

event The event that occurred.

□ Called by the TypedEventBus service to notify the listener of an unhandled event.

157.9.8 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 TypedEventConstants.TYPED_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);
```

Concurrency Thread-safe

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

topic The topic to which the event was sent

event The event that occurred.

□ Called by the TypedEventBus service to notify the listener of an event.

157.10 org.osgi.service.typedevent.annotations

Typed Event Annotations Package Version 1.0.

This package contains annotations that can be used to require the Typed Event implementation.

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

157.10.1 Summary

- `RequireTypedEvent` - This annotation can be used to require the Typed Event implementation.

157.10.2 @RequireTypedEvent

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

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

Since 1.0

DRAFT
Licensed under the Eclipse Foundation Specification License v1.0. Copyright © Contributors to the Eclipse Foundation.

Retention CLASS

Target TYPE, PACKAGE

157.11 org.osgi.service.typedevent.monitor

Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT

Typed 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.typedevent.monitor; version="[1.0,2.0)"`

Example import for providers implementing the API in this package:

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

157.11.1 Summary

- `MonitorEvent` - A monitoring event.
- `TypedEventMonitor` - The EventMonitor service can be used to monitor the events that are sent using the EventBus, and that are received from remote EventBus instances

157.11.2 public class MonitorEvent

A monitoring event.

Provider Type Consumers of this API must not implement this type

157.11.2.1 public Map<String, Object> eventData

The Data from the Event in Map form

157.11.2.2 public Instant publicationTime

The time at which the event was published

157.11.2.3 public String topic

The Event Topic

157.11.2.4 public MonitorEvent()

157.11.3 public interface TypedEventMonitor

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

Concurrency Thread-safe

Provider Type Consumers of this API must not implement this type

157.11.3.1 public int configureHistoryStorage(String topicFilter, int minRequired, int maxRequired)

topicFilter the topic filter

minRequired the minimum number of historical events to keep available for this filter

maxRequired the maximum number of historical events to keep available for this filter

□ Configure history storage for a given topic filter.

Minimum storage settings may only be set for exact matches. It is an error to use a filter containing wildcards with a non-zero minimum history requirement.

If a minimum storage requirement is set then the Typed Events implementation must guarantee sufficient storage space to hold those events. If, after accounting for all other pre-existing minium-mum storage requirements, there is insuffcient storage left for this new configuration then an IllegalStateException must be thrown.

Returns An int indicating the number of events that can be kept for this topic given the current configuration. This will always be at least `minRequired` and at most `maxRequired`.

Throws `NullPointerException`– if the topic filter is `null`

`IllegalArgumentException`– if:

- The topic filter contains invalid syntax
- `minRequired` or `maxRequired` are less than `0`.
- The topic filter contains wildcard(s) and `minRequired` is not `0`.
- `minRequired` is greater than `maxRequired`

`IllegalStateException`– if there is insufficient available space to provide the additional `minRequired` stored events.

Since 1.1

157.11.3.2 public Map<String, Map.Entry<Integer, Integer>> getConfiguredHistoryStorage()

□ Get the configured history storage for the Typed Events implementation.

The returned Map uses topic filter strings as keys. These filter strings may contain wildcards. If multiple filter strings are able to match then the most specific match applies with the following ordering:

1. An exact topic match
2. An exact match of the parent topic and a single level wildcard as the final token
3. An exact match of the parent topic and multi-level wildcard as the final token

This ordering is applied recursively starting with the first topic token until only one candidate remains. The keys in the returned map are ordered such that the first encountered key which matches a given topic name is the configuration that will apply to that topic name.

The value associated with each key is an Entry where the key is the minimum required number of stored events for the topic and the value is the maximum number of events that will be stored.

Returns The configured history storage

Since 1.1

157.11.3.3 public Map.Entry<Integer, Integer> getConfiguredHistoryStorage(String topicFilter)

topicFilter the topic filter

□ Get the configured history storage for a given topic filter. This method looks for an exact match in the history configuration. If no configuration is found for the supplied topic filter then `null` is returned.

Returns An Entry where the key is the minimum required number of stored events for the topic and the value is the maximum number of events that will be stored. If no configuration is set for the topic filter then `null` will be returned.

Throws `NullPointerException`– if the topic filter is `null`

`IllegalArgumentException`– if the topic filter contains invalid syntax

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation

Since 1.1

157.11.3.4 public Map.Entry<Integer, Integer> getEffectiveHistoryStorage(String topicName)

topicName the topic name

- Get the history storage rule that applies to a given topic name. This method takes into account the necessary precedence rules to find the correct configuration for the named method, and so will never return null.

Returns An Entry where the key is the minimum required number of stored events for the topic and the value is the maximum number of events that will be stored. If no configuration is set for the topic filter then an entry with key and value set to zero will be returned.

Throws NullPointerException– if the topic name is null

IllegalArgumentException– if the topic name contains invalid syntax or wildcards

Since 1.1

157.11.3.5 public long getMaximumEventStorage()

- Get an estimate of the maximum number of historic events that can be stored by the TypedEvent implementation. If there is no fixed limit then -1 is returned. If no history storage is supported then zero is returned.

Returns The maximum number of historic events that can be stored.

Since 1.1

157.11.3.6 public PushStream<MonitorEvent> monitorEvents()

- Get a stream of events, starting now.

Returns A stream of event data

157.11.3.7 public PushStream<MonitorEvent> monitorEvents(int history)

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.

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

Logically equivalent to monitorEvents(history, false).

Returns A stream of event data

157.11.3.8 public PushStream<MonitorEvent> monitorEvents(int history, boolean historyOnly)

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.

historyOnly If true then the returned stream will be closed as soon as the available history has been delivered

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

Returns A stream of event data

Since 1.1

157.11.3.9 public PushStream<MonitorEvent> monitorEvents(Instant history)

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

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

Logically equivalent to monitorEvents(history, false).

Returns A stream of event data

DRAFT
Licensed under the Eclipse Foundation Specification License v1.0. Copyright © Contributors to the Eclipse Foundation.

157.11.3.10 **public PushStream<MonitorEvent> monitorEvents(Instant history, boolean historyOnly)**

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

historyOnly If true then the returned stream will be closed as soon as the available history has been delivered

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

Returns A stream of event data

Since 1.1

157.11.3.11 **public void removeHistoryStorage(String topicFilter)**

topicFilter the topic filter

□ Delete history storage configuration for a given topic filter.

Throws NullPointerException– if the topic filter is null

IllegalArgumentException– if the topic filter contains invalid syntax

Since 1.1

157.11.3.12 **public Predicate<String> topicFilterMatches(String topicFilter)**

topicFilter The topic filter to match against

□ Get a Predicate which will match the supplied topic filter against a topic name.

Returns A predicate that will return true if the topic name being tested matches the supplied topic filter.

Throws NullPointerException– if the topic filter is null

IllegalArgumentException– if the topic filter contains invalid syntax

Since 1.1

157.11.3.13 **public boolean topicFilterMatches(String topicName, String topicFilter)**

topicName The topic name to match against

topicFilter The topic filter to match against

□ Test the supplied topic filter against the supplied topic name.

Returns A predicate that will return true if the topic name being tested matches the supplied topic filter.

Throws NullPointerException– if the topic filter is null

IllegalArgumentException– if the topic filter or topic name contain invalid syntax

Since 1.1

157.12 org.osgi.service.typedevent.propertytypes

Typed 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.typedevent.propertytypes; version="[1.0,2.0)"
```

DRAFT
Licensed under the Eclipse Foundation Specification License v1.0, Copyright © Contributors to the Eclipse Foundation

157.12.1 Summary

- EventFilter - Component Property Type for the TypedEventConstants.TYPED_EVENT_FILTER service property of an Event Handler service.
- EventTopics - Component Property Type for the TypedEventConstants.TYPED_EVENT_TOPICS service property of a TypedEventHandler, UntypedEventHandler or UnhandledEventHandler service.
- EventType - Component Property Type for the TypedEventConstants.TYPED_EVENT_TYPE service property of an TypedEventHandler service.

157.12.2 @EventFilter

Component Property Type for the TypedEventConstants.TYPED_EVENT_FILTER service property of an Event Handler service.

This annotation can be used on an TypedEventHandler, UntypedEventHandler or UnhandledEventHandler component to declare the value of the TypedEventConstants.TYPED_EVENT_FILTER service property.

See Also Component Property Types

Retention CLASS

Target TYPE

157.12.2.1 String value

□ Service property specifying the event filter for a TypedEventHandler or UntypedEventHandler service.

Returns The event filter.

See Also TypedEventConstants.TYPED_EVENT_FILTER

157.12.3 @EventTopics

Component Property Type for the TypedEventConstants.TYPED_EVENT_TOPICS service property of a TypedEventHandler, UntypedEventHandler or UnhandledEventHandler service.

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

See Also Component Property Types

Retention CLASS

Target TYPE

157.12.3.1 String[] value

□ Service property specifying the Event topics of interest to an TypedEventHandler, UntypedEventHandler or UnhandledEventHandler service.

Returns The event topics.

See Also TypedEventConstants.TYPED_EVENT_TOPICS

157.12.4 @EventType

Component Property Type for the TypedEventConstants.TYPED_EVENT_TYPE service property of an TypedEventHandler service.

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

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

See Also Component Property Types

Retention CLASS

Target TYPE

157.12.4.1 Class<?> value

□ Service property specifying the EventType for a TypedEventHandler service.

Returns The event filter.

See Also TypedEventConstants.TYPED_EVENT_TYPE

157.13 References

[1] *Java Records*
https://openjdk.org/jeps/395

157.14 Changes

The update to version 1.1 includes numerous new features and usability enhancements:

- Event objects may now be [1] *Java Records*
- Event Handler services may now use *Single-Level Wildcards* on page 13 when registering to receive event data.
- The TYPED_EVENT_HISTORY service property can be used to request that historical events are delivered to newly registered event handler services.
- Unhandled event handler services may now use TYPED_EVENT_TOPICS and TYPED_EVENT_FILTER service properties to filter the events that they receive.
- A TypedEventPublisher object can be created to optimise sending repeated events to the same topic.
- The TypedEventMonitor service can be requested to provide only historical events, terminating the event stream once all historical data has been replayed.
- Historical event storage can now be configured at runtime on a per-topic basis using the TypedEventMonitor.

Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.
DRAFT

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT
Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

Licensed under the Eclipse Foundation Specification License – v1.0. Copyright © Contributors to the Eclipse Foundation.

DRAFT

End Of Document