Platform Events (Beta) Developer Guide

Version 39.0, Spring '17

@salesforcedocs

Last updated: March 10, 2017

© Copyright 2000–2017 salesforce.com, inc. All rights reserved. Salesforce is a registered trademark of salesforce.com, inc.,
as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

Chapter 1: Delivering Custom Notifications with Platform Events (Beta) . . . 1
Event-Driven Software Architecture . . . 2
Enterprise Messaging Platform Events . . . 3
What Is the Difference Between the Salesforce Events? . . . 3

Chapter 2: Defining Your Platform Event . . . 4
Platform Event Fields . . . 5
Migrate Platform Event Definitions with Metadata API . . . 6
Platform Event Considerations . . . 7

Chapter 3: Publish Platform Events . . . 8
Publish Event Messages Using Apex . . . 9
Publish Event Messages Using Salesforce APIs . . . 9

Chapter 4: Subscribing to Platform Events . . . 12
Subscribe to Platform Event Notifications with Apex Triggers . . . 13
    Refire Event Triggers with EventBus.RetryableException . . . 13
Subscribe to Platform Event Notifications with CometD . . . 14
Obtain Event Subscribers . . . 15

Chapter 5: Platform Event API Considerations and Testing . . . 16
Apex, API, and SOQL Considerations for Platform Events . . . 17
Test Your Platform Event in Apex . . . 18

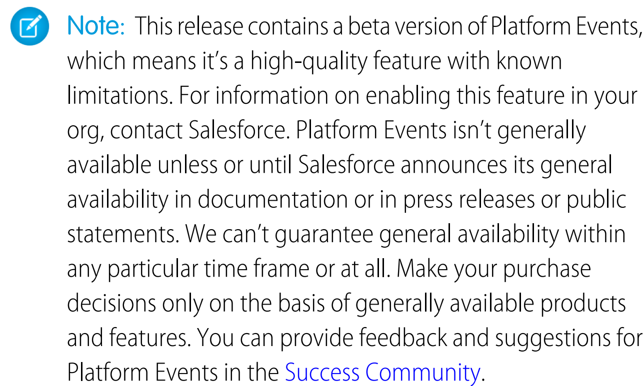
Chapter 6: Reference . . . 19
Platform Event Limits . . . 20
EventBusSubscriber . . . 20
EventBus Class . . . 22
    EventBus Methods . . . 22
TriggerContext Class . . . 24
    TriggerContext Properties . . . 24
    TriggerContext Methods . . . 25

CHAPTER 1 Delivering Custom Notifications with Platform Events (Beta)

In this chapter ...

- Event-Driven Software Architecture
- Enterprise Messaging Platform Events
- What Is the Difference Between the Salesforce Events?

Use platform events to deliver secure and scalable custom notifications within Salesforce or from external sources. Define fields to customize your platform event. Your custom platform event determines the event data that the Force.com platform can produce or consume.



Note: This release contains a beta version of Platform Events, which means it's a high-quality feature with known limitations. For information on enabling this feature in your org, contact Salesforce. Platform Events isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. We can't guarantee general availability within any particular time frame or at all. Make your purchase decisions only on the basis of generally available products and features. You can provide feedback and suggestions for Platform Events in the Success Community.

Platform events are part of Salesforce's enterprise messaging platform. This platform provides an event-driven messaging architecture to enable apps to communicate inside and outside of Salesforce. Before diving into platform events, first take a look at what an event-based software system is.

EDITIONS

Platform Events is available in both Lightning Experience and Salesforce Classic. The definition of platform events is available in Salesforce Classic only.

Available in: **Performance**, **Unlimited**, **Enterprise**, and **Developer** Editions

Event-Driven Software Architecture

An event-driven (or message-driven) software architecture consists of event producers, event consumers, and channels.The architecture is suitable for large distributed systems because it decouples event producers from event consumers, thereby simplifying the communication model in connected systems.

Event
: A change in state that is meaningful in a business process. For example, a placement of a purchase order is a meaningful event because the order fulfillment center requires notification to process the order. Or a change in a refrigerator's temperature can indicate that it needs service.

Event message
: A message that contains data about the event. Also known as an event notification.

Event producer
: The publisher of an event message over a channel.

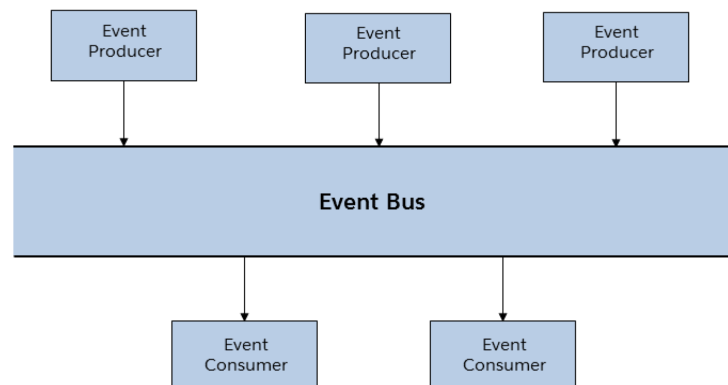
Channel
: A conduit in which an event producer transmits a message. Event consumers subscribe to the channel to receive messages. Also referred to as event bus in Salesforce.

Event consumer
: A subscriber to a channel that receives messages from the channel.

Systems in request-response communication models make a request to a web service or database to obtain information about a certain state. The sender of the request establishes a connection to the service and depends on the availability of the service.

In comparison, systems in an event-based model obtain information and can react to it in near real time when the event occurs. Also, producers and consumers don't have dependencies on each other. Event producers don't know the consumers that receive the events. Any number of consumers can receive and react to the same events. The only dependency between producers and consumers is the semantic of the message content.

The following diagram illustrates an event-based software architecture system.



Enterprise Messaging Platform Events

The Salesforce enterprise messaging platform is event-based and offers the benefits of event-driven software architectures. Platform events are the event messages (or notifications) that your apps send and receive to take further actions. Platform events simplify the process of communicating changes and responding to them without writing complex logic. Publishers and subscribers communicate with each other through events. Multiple subscribers can listen to the same event and carry out different actions.

You can customize the schema of platform events to define which data types to send in a message.

You can publish and consume platform events by using Apex or an API. Platform events integrate with the Salesforce platform through Apex triggers. Triggers are the event consumers on the Salesforce platform that listen to event messages. Whether an external app through the API or a native Force.com app through Apex published the event message, a trigger on that event gets fired. Triggers run the actions in response to the event notifications.

For example, a software system monitoring a printer makes an API call to publish a custom event when the ink is low. The printer event message contains custom fields for the printer model, serial number, and ink level. After the printer sends the event message, an Apex trigger is fired in Salesforce. The trigger creates a Case record to place an order for a new cartridge.

As an alternative to using Apex triggers, external apps can listen to event notifications by subscribing to a channel through CometD.

What Is the Difference Between the Salesforce Events?

Salesforce offers various features that use events. Except for Platform Events and Streaming API generic events, most of these events are notifications within Salesforce or calendar items.

The following is a partial list of the types of events provided.

Platform Events
: Platform events enable you to deliver secure, scalable, and customizable event notifications within Salesforce or from external sources. Platform event fields are defined in Salesforce and determine the data that you send and receive. Apps can publish and subscribe to platform events on the Force.com Platform using Apex or in external systems using CometD.

Streaming API Events
: Streaming API provides two types of events that you can publish and subscribe to: PushTopic and generic. PushTopic events track field changes in Salesforce records and are tied to Salesforce records. Generic events contain arbitrary payloads. Both event types don't provide the level of granular customization that platform events offer. You can send a custom payload with a generic event, but you can't define the data as fields. You can't define those types of events in Salesforce, and you can't use them in Apex triggers.

Event Monitoring
: Event monitoring enables admins to track user activity and the org's performance. In this context, events are actions that users perform, such as logins and exporting reports. The events are internal and logged by Salesforce. You can query the events, but you can't publish the events or subscribe to them in real time.

Transaction Security Policies
: Transaction security policies evaluate user activity, such as logins and data exports, and trigger actions in real time. When a policy is triggered, notifications are sent through email or in-app notifications. Actions can be standard actions, such as blocking an operation, or a custom action defined in Apex.

Calendar Events
: Calendar events in Salesforce are appointments and meetings you can create and view in the user interface. In the SOAP API, the Event object represents a calendar event. Those events are calendar items and not notifications that software systems send.

This guide focuses on Platform Events only.

CHAPTER 2 Defining Your Platform Event

In this chapter ...

- Platform Event Fields
- Migrate Platform Event Definitions with Metadata API
- Platform Event Considerations

Platform events are sObjects, similar to custom objects but with some limitations. Event notifications are instances of platform events. Unlike sObjects, you can't update event records. You also can't view the event records in the user interface. When you delete a platform event definition, it's permanently deleted.

USER PERMISSIONS

To create and edit platform event definitions:

- "Customize Application"

Platform Event Fields

To customize a platform event, create an event and add fields.

To define a platform event in the Salesforce user interface, from Setup, enter *`Platform Events`* in the `Quick Find` box, then select **Platform Events**.

Standard Fields

Platform events include standard fields. These fields appear on the New Platform Event page.

Field	Description
`Label`	Name used to refer to your platform event in a user interface page.
`Plural Label`	Plural name of the platform event.
`Starts with a vowel sound`	If it's appropriate for your org's default language, indicate whether the label is preceded by "an" instead of "a."
`Object Name`	Unique name used to refer to the platform event when using the API. In managed packages, this name prevents naming conflicts with package installations. Use only alphanumeric characters and underscores. The name must begin with a letter and have no spaces. It cannot end with an underscore nor have two consecutive underscores.
`Description`	Optional description of the object. A meaningful description helps you remember the differences between your events when you are viewing them in a list.
`Deployment Status`	Indicates whether the platform event is visible to other users.

Custom Fields

In addition to the standard fields, add custom fields to customize your event. Platform event custom fields support only these field types.

- Checkbox
- Date
- Date/Time
- Number
- Text
- Text Area (Long)

`ReplayId` System Field

The `ReplayId` number field identifies each event record and is populated by the system. Each replay ID is guaranteed to be higher than the ID of the previous event but not necessarily contiguous for consecutive events. The ID is unique for the org and the channel and is used to replay past events.

API Name Suffix for Platform Events

When you create a platform event, the system appends `__e` to create the API name of the event. For example, if you create an event with the object name `Low Ink`, the API name is `Low_Ink__e`. The API name is used whenever you refer to the event programmatically, for example, in Apex.

Migrate Platform Event Definitions with Metadata API

Deploy and retrieve platform event definitions from your sandbox and production org as part of your app's development lifecycle.

The CustomObject metadata type represents platform events.

Platform event names are appended with `__e`. The file that contains the platform event definition has the suffix `.object`. Platform events are stored in the `objects` folder.

Here is a definition of a sample platform event with one text field.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <deploymentStatus>Deployed</deploymentStatus>
    <fields>
        <fullName>DemoTextField__c</fullName>
        <externalId>false</externalId>
        <isFilteringDisabled>false</isFilteringDisabled>
        <isNameField>false</isNameField>
        <isSortingDisabled>false</isSortingDisabled>
        <label>DemoTextField</label>
        <length>16</length>
        <required>false</required>
        <type>Text</type>
        <unique>false</unique>
    </fields>
    <label>DemoPlatformEvent</label>
    <pluralLabel>DemoPlatformEvents</pluralLabel>
</CustomObject>
```

This package.xml manifest file references the previous event definition. The name of the referenced event is `DemoPlatformEvent__e`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>DemoPlatformEvent__e</members>
        <name>CustomObject</name>
    </types>
    <version>39.0</version>
</Package>
```

SEE ALSO:

Metadata API Developer Guide

Platform Event Considerations

Take note of the considerations when defining platform events.

Permanent Deletion of Event Definitions

When you delete an event definition, it's permanently removed and can't be restored. Before you delete the event definition, delete the associated triggers. Published events that use the definition are also deleted.

Renaming Event Objects

Before you rename an event, delete the associated triggers. If the event name is modified after clients have subscribed to notifications for this event, the subscribed clients must resubscribe to the updated topic. To resubscribe to the new event, add your trigger for the renamed event object.

No Associated Tab

Platform events don't have an associated tab because you can't view event records in the Salesforce user interface.

No Record Page Support in Lightning App Builder

When creating a record page in Lightning App Builder, platform events that you defined show up in the list of objects for the page. However, you can't create a Lightning record page for platform events because event records aren't available in the user interface.

CHAPTER 3 Publish Platform Events

In this chapter ...

- Publish Event Messages Using Apex
- Publish Event Messages Using Salesforce APIs

After a platform event has been defined in your Salesforce org, you can publish event messages from a Force.com app or an external app using Apex or Salesforce APIs.

Publish Event Messages Using Apex

Use Apex to publish event messages from a Force.com app.

To publish event messages, call the `EventBus.publish` method. For example, if you've defined a custom platform event called `Low Ink`, reference this event type as `Low_Ink__e`. Next create instances of this event and pass them to the Apex method.

This example creates two events of type `Low_Ink__e`, publishes them, and then checks whether the publishing was successful or errors were encountered. The example assumes that the `Low Ink` platform event is defined in your org.

```
List<Low_Ink__e> inkEvents = new List<Low_Ink__e>();
inkEvents.add(new Low_Ink__e(Printer_Model__c='XZO-5', Serial_Number__c='12345',
              Ink_Percentage__c=0.2));
inkEvents.add(new Low_Ink__e(Printer_Model__c='MN-123', Serial_Number__c='10013',
              Ink_Percentage__c=0.15));


// Call method to publish events
List<Database.SaveResult> results = EventBus.publish(inkEvents);

// Inspect publishing result for each event
for (Database.SaveResult sr : results) {
    if (sr.isSuccess()) {
        System.debug('Successfully published event.');
    } else {
        for(Database.Error err : sr.getErrors()) {
            System.debug('Error returned: ' +
                         err.getStatusCode() +
                         err.getMessage());
        }
    }
}
```

When you publish events from Apex, they're inserted synchronously. Because event publishing is equivalent to a DML insert operation, DML limits apply.

Publish Event Messages Using Salesforce APIs

External apps use an API to publish platform event messages.

Publish events by creating records of your event in the same way that you insert sObjects. You can use any Salesforce API to create platform events, such as SOAP API, REST API, or Bulk API.

For example, if you've defined a platform event named `Low Ink`, publish event notifications by inserting `Low_Ink__e` records. This example creates one event of type `Low_Ink__e` in REST API.

REST endpoint:

```
/services/data/v37.0/sobjects/Low_Ink__e/
```

Request body:

```
{
   "Printer_Model__c" : "XZO-5"
}
```

After the platform event record is created, the REST response looks like this output. Headers are deleted for brevity.

```
HTTP/1.1 201 Created

{
   "id" : "e00xx000000000B",
   "success" : true,
   "errors" : [ ]
}
```

For more information, see the Force.com REST API Developer Guide.

This example shows the SOAP message (using Partner API) of a request to create three platform events in one call. Each event has one custom field named `Printer_Model__c`.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="urn:sobject.partner.soap.sforce.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="urn:partner.soap.sforce.com">
<SOAP-ENV:Header>
    <ns2:SessionHeader>
        <ns2:sessionId>00DR00000001fWV!AQMAQOshATCQ4fBaYFOTrHVixfEO6l...</ns2:sessionId>
    </ns2:SessionHeader>
    <ns2:CallOptions>
        <ns2:client>Workbench/34.0.12i</ns2:client>
        <ns2:defaultNamespace xsi:nil="true"/>
        <ns2:returnFieldDataTypes xsi:nil="true"/>
    </ns2:CallOptions>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <ns2:create>
        <ns2:sObjects>
            <ns1:type>Low_Ink__e</ns1:type>
            <ns1:fieldsToNull xsi:nil="true"/>
            <ns1:Id xsi:nil="true"/>
            <Printer_Model__c>XZO-600</Printer_Model__c>
        </ns2:sObjects>
        <ns2:sObjects>
            <ns1:type>Low_Ink__e</ns1:type>
            <ns1:fieldsToNull xsi:nil="true"/>
            <ns1:Id xsi:nil="true"/>
            <Printer_Model__c>XYZ-100</Printer_Model__c>
        </ns2:sObjects>
        <ns2:sObjects>
            <ns1:type>Low_Ink__e</ns1:type>
            <ns1:fieldsToNull xsi:nil="true"/>
            <ns1:Id xsi:nil="true"/>
            <Printer_Model__c>XYZ-9000</Printer_Model__c>
        </ns2:sObjects>
    </ns2:create>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The response of the Partner SOAP API request looks something like the following. Headers are deleted for brevity.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="urn:partner.soap.sforce.com">
<soapenv:Header>
 ...
</soapenv:Header>
<soapenv:Body>
    <createResponse>
        <result>
            <id>e00xx000000000F</id>
            <success>true</success>
        </result>
        <result>
            <id>e00xx000000000G</id>
            <success>true</success>
        </result>
        <result>
            <id>e00xx000000000H</id>
            <success>true</success>
        </result>
    </createResponse>
</soapenv:Body>
</soapenv:Envelope>
```

For more information about creating records, see the `create()` call in the SOAP API Developer Guide.

CHAPTER 4 Subscribing to Platform Events

In this chapter ...

- Subscribe to Platform Event Notifications with Apex Triggers
- Subscribe to Platform Event Notifications with CometD
- Obtain Event Subscribers

Receive platform events in Apex triggers or in CometD clients.

Subscribe to Platform Event Notifications with Apex Triggers

Use Apex triggers to subscribe to events. You can receive event notifications in triggers regardless of how they were published—through Apex or APIs. Triggers provide an autosubscription mechanism. No need to explicitly create and listen to a channel in Apex.

An Apex trigger processes platform event notifications sequentially in the order they're received. The order of events is based on the event replay ID. An Apex trigger can receive a batch of events at once. In this case, the order of events is preserved within each batch.

To subscribe to event notifications, write an `after insert` trigger on the event object type. The `after insert` trigger event corresponds to the time after a platform event is published. After an event message is published, the `after insert` trigger is fired.

This example shows a trigger for the `Low Ink` event. It iterates through each event and checks a field value. The trigger inspects each received notification and gets the printer model from the notification. If the printer model matches a certain value, other business logic is executed. For example, the trigger creates a case to order a new cartridge for this printer model.

```
// Trigger for catching Low_Ink events.
trigger LowInkTrigger on Low_Ink__e (after insert) {
    // List to hold all cases to be created.
    List<Case> cases = new List<Case>();

    // Get user Id for case owner
    User usr = [SELECT Id FROM User WHERE Name='Admin User' LIMIT 1];

    // Iterate through each notification.
    for (Low_Ink__e event : Trigger.New) {
        System.debug('Printer model: ' + event.Printer_Model__c);
        if (event.Printer_Model__c == 'MN-123') {
            // Create Case to order new printer cartridge.
            Case cs = new Case();
            cs.Priority = 'Medium';
            cs.Subject = 'Order new ink cartridge for SN ' + event.Serial_Number__c;
            cs.OwnerId = usr.Id;
            cases.add(cs);
        }
    }

    // Insert all cases corresponding to events received.
    insert cases;
}
```

Note: If you create a Salesforce record with an `ownerId` field in the trigger, explicitly set the owner ID. For cases and leads, you can alternatively use assignment rules to set the owner. See Apex, API, and SOQL Considerations for Platform Events.

Refire Event Triggers with EventBus.RetryableException

Refiring an event trigger gives you another chance to process event notifications. Refiring a trigger is helpful when a transient error occurs or when waiting for a condition to change. Refire a trigger if the error or condition is external to the event records and is likely to go away later.

Refire Event Triggers with `EventBus.RetryableException`

Refiring an event trigger gives you another chance to process event notifications. Refiring a trigger is helpful when a transient error occurs or when waiting for a condition to change. Refire a trigger if the error or condition is external to the event records and is likely to go away later.

For example, a trigger adds a related record to a master record if a field on the master record equals a certain value. It is possible that in a subsequent try, the field value changes and the trigger can perform the operation.

To refire the event trigger, throw `EventBus.RetryableException`. The event is resent after a small delay. The delay increases with each subsequent retry. A resent event has the same field values as the original event, but the batch sizes of the events can differ. For example, the initial trigger can receive events with replay ID 10 to 20. The resent batch can be larger, containing events with replay ID 10 to 40.

This example is a skeletal trigger that gives you an idea of how to throw `EventBus.RetryableException`. The trigger uses an `if` statement to check whether a certain condition is true. Alternatively, you can use a try-catch block and throw `EventBus.RetryableException` in the catch block.

```
trigger ResendEventsTrigger on Low_Ink__e (after insert) {
    if (condition == true) {
        // Process platform events.
    } else {
        // Condition isn't met, so try again later.
        throw new EventBus.RetryableException();
    }
}
```

The previous example refires the trigger every time the condition isn't met by throwing `EventBus.RetryableException`. But what if you want to place a limit on how many times you want to refire the trigger? For example, refire the trigger for up to three times only. This is when the `EventBus.TriggerContext` class comes in handy. The `EventBus.TriggerContext.currentContext()` returns an instance of the `EventBus.TriggerContext` class that contains information about the currently executing trigger. Calling the `retries` property on this instance gets the number of times the trigger was refired.

```
trigger ResendEventsTrigger on Low_Ink__e (after insert) {
    if (condition == true) {
        // Process platform events.
    } else {
    // Ensure we don't refire the trigger more than 3 times
    if (EventBus.TriggerContext.currentContext().retries < 4) {
        // Condition isn't met, so try again later.
        throw new EventBus.RetryableException(
                      'Condition is not met, so retrying the trigger again.');
    } else {
        // Trigger was refired enough times so give up and
        // resort to alternative action.
        // For example, send email to user.
    }
}
```

Subscribe to Platform Event Notifications with CometD

Use EMP Connector to receive platform events in an external Java app. EMP Connector connects to CometD and hides the complexity of subscribing to events.

Salesforce sends platform events to CometD clients, including EMP Connector, sequentially in the order they're received. The order of event notifications is based on the replay ID of events.

The process of subscribing to platform event notifications through CometD is similar to subscribing to PushTopics or generic events. The only difference is the channel name. Here is the format of the platform event topic (channel) name.

```
/event/<EventName>__e
```

For example, if you have a platform event named `Low Ink`, provide this channel name when subscribing.

```
/event/Low_Ink__e
```

Ensure that your API client uses version 37.0 or later of the CometD endpoint.

```
/cometd/39.0
```

The message of a delivered platform event looks similar to the following example for `Low Ink` events.

```
{
  "clientId": "123qiewfarn041rn29pf37awjr",
  "data": {
    "payload": {
      "CreatedById": "005D0000001WHiZ",
      "ReplayId": null,
      "CreatedDate": "2016-12-14T20:08:19Z",
      "Printer_Model__c": "MN-123",
      "Id": null,
      "Serial_Number__c": "10013",
      "Ink_Percentage__c": 0.15
    },
    "event": {
      "schema": "_c1_d37defc8cfbf44229cbd4fe82167be19",
      "replayId": 7
    }
  },
  "channel": "/event/Low_Ink__e"
}
```

Use EMP Connector to receive delivered events. The connector subscribes to streaming events and platform events in the same way—only the topic name is different. See Example: Subscribe to and Replay Events Using a Java Client in the Streaming API Developer Guide. For the *`topic`* argument, provide `/event/Low_Ink__e`. The topic name value is based on the example event `Low Ink`.

Add custom logic to your client to perform some operations after a platform event notification is received. For example, the client can create a request to order a new cartridge for this printer model.

Obtain Event Subscribers

View a list of all triggers that are subscribed to a platform event by using the Salesforce user interface or the API.

Note: CometD subscribers to a platform event channel aren't currently exposed in the user interface or the API.

View all the triggers that are subscribed to a platform event on the event's definition page.

1. From Setup, enter *`Platform Events`* in the `Quick Find` box, then select **Platform Events**.
2. Click your event's name.

On the event's definition page, the Subscriptions related list shows all the triggers that are subscribed to platform events. The list shows the platform event's last replay ID that each subscription processed and whether errors occurred.

Alternatively, you can obtain the same subscriber information by querying the EventBusSubscriber object. See EventBusSubscriber.

CHAPTER 5 Platform Event API Considerations and Testing

In this chapter ...

- Apex, API, and SOQL Considerations for Platform Events
- Test Your Platform Event in Apex

Learn about special behaviors of platform events in Apex and Salesforce APIs and how to test them.

Apex, API, and SOQL Considerations for Platform Events

Be familiar with the considerations when publishing and subscribing to platform events.

Only `after insert` Triggers Are Supported

Only `after insert` triggers are supported for platform events because event notifications can't be updated. They're only inserted (published).

Infinite Trigger Loop and Limits

Be careful when publishing events from triggers because you could get into an infinite trigger loop and exceed daily event limits. For example, if you publish an event from a trigger that's associated with the same event object, the trigger is fired in an infinite loop.

Apex DML Limits for Publishing Events

Each `EventBus.publish` method call is considered a DML statement, and DML limits apply.

Platform Event Triggers: `ownerId` Fields of New Records

If you create Salesforce records in platform event triggers, set the `ownerId` field in those records explicitly to the appropriate user. Platform event triggers run under the Automated Process entity. If you don't set the `ownerId` field on records that contain this field, the system sets the default value of `Automated Process`. This example explicitly populates the `ownerId` field for an opportunity with an ID obtained from another record.

```
Opportunity newOpp = new Opportunity(
    AccountId = acc.Id,
    StageName = 'Qualification',
    Name = 'A ' + customerOrder.Product_Name__c + ' opportunity for ' + acc.name,
    CloseDate = Date.today().addDays(7),
    OwnerId = customerOrder.createdById);
```

For cases and leads, you can alternatively use assignment rules for setting the owner. See AssignmentRuleHeader for the SOAP API or Setting DML Options for Apex.

Debug Logs for Platform Event Triggers

Debug logs for platform event triggers are created by a process called "Automated Process" and are separate from their corresponding Apex code logs. The debug logs aren't available in the Developer Console's Log tab. One exception is Apex tests, which include debug logging for event triggers in the same test execution log. To collect platform event trigger logs, add a trace flag entry for the Automated Process user on the Debug Logs page in Setup. To collect logs in the Debug Logs page for your Apex code that publishes the events, add another trace flag entry for your user.

API Request Limits for Publishing Events

Because platform events are published by inserting the event sObjects, API request limits apply. For more information, see API Request Limits in the Salesforce Limits Quick Reference Guide.

Replaying Past Events

You can replay platform events that were sent in the past 24 hours. You can replay platform events through the API but not Apex. The process of replaying platform events is the same as for other Streaming API events. For more information, see the following resources.

- Example: Subscribe to and Replay Events Using a Java Client
- Example: Subscribe to and Replay Events Using a Visualforce Page
- Streaming Replay Client Extensions for Java and JavaScript on GitHub

No SOQL Support

You can't query event notifications using SOQL.

Test Your Platform Event in Apex

Use `Test.startTest()` and `Test.stopTest()` to test your platform event in Apex.

Create test event objects, and publish them after the `Test.startTest()` statement. Then call the `Test.stopTest()` statement to publish the test events. Include your validations after the `Test.stopTest()` statement.

```
Test.startTest();
// Create test events & publish them
Test.stopTest();
// Perform validation here
```

This sample test class creates one Low_Ink__e event in a test method. After `Test.stopTest()`, a SOQL query verifies that the associated trigger was fired. The trigger creates a case. The case subject contains the printer serial number. This example requires the Low_Ink__e event to be defined in the org.

```
@isTest
public class PlatformEventTest {
    @isTest static void test1() {
        Test.startTest();

        // Create a test event and publish it
        Low_Ink__e inkEvent = new Low_Ink__e(Printer_Model__c='MN-123',
                                             Serial_Number__c='10013',
                                             Ink_Percentage__c=0.15);
        EventBus.publish(inkEvent);

        Test.stopTest();

        // Perform validation here
        // Get a case whose subject contains the serial number of the test event.
        // This case was created by a trigger.
        List<Case> cases = [SELECT Id FROM Case WHERE
                            Subject LIKE '%:inkEvent.Serial_Number__c'];
        // Validate that this case was found
        System.assertEquals(1, cases.size());
    }

}
```

CHAPTER 6 Reference

In this chapter ...

- Platform Event Limits
- EventBusSubscriber
- EventBus Class
- TriggerContext Class

The reference documentation for platform events covers limits, an API object, and Apex methods.

Platform Event Limits

The following limits apply to publishing platform events, event delivery in CometD clients, and platform event definitions.

Description	Performance and Unlimited Editions	Enterprise Edition	All other editions
Maximum number of events published per hour (Events can be published using Apex or APIs.)	100,000	100,000	1,000
Maximum number of events delivered to CometD clients within a 24–hour period	250,000	100,000	10,000
Maximum number of platform event definitions that can be created in an org	100	50	5

If your application exceeds these limits, or you have scenarios that require creating and publishing more events, contact Salesforce to request a higher limit.

EventBusSubscriber

Represents a trigger that is subscribed to a platform event.

Supported Calls

`query()`

Special Access Rules

EventBusSubscriber is read only and can only be queried.

Fields

Field	Details
`ExternalId`	**Type** string **Properties** Filter, Group, Nillable, Sort **Description** The ID of the subscriber. For example, the trigger ID.
`Name`	**Type** string

Field	Details
	Properties Filter, Group, Nillable, Sort **Description** The name of the subscribed item, such as the trigger name.
`Position`	**Type** int **Properties** Filter, Group, Nillable, Sort **Description** The replay ID of the last event that the subscriber processed.
`Status`	**Type** picklist **Properties** Filter, Group, Nillable, Restricted picklist, Sort **Description** Indicates the status of the subscriber. Can be one of the following values: • `Running`—The subscriber is actively listening to events. • `Suspended`—The subscriber is disconnected and can't receive events due to lack of permissions. • `Expired`—The subscriber's connection expired. In rare cases, subscriptions can expire if they're inactive for an extended period of time. • `Error`—The subscription encountered an error and has been disconnected.
`Tip`	**Type** int **Properties** Filter, Group, Nillable, Sort **Description** The replay ID of the last published event.
`Topic`	**Type** string **Properties** Filter, Group, Nillable, Sort **Description** The name of the subscription channel that corresponds to a platform event. The topic name is the event name appended with `__e`, such as `MyEvent__e`. The topic is the channel that the subscriber is subscribed to.

Field	Details
Type	**Type** string **Properties** Filter, Group, Nillable, Sort **Description** The subscriber type. Can be one of the following values: • ApexTrigger • Process—Reserved for future use.

Usage

Use EventBusSubscriber to query details about subscribers to a platform event. You can get all subscribers for a particular event by filtering on the Topic field, as follows.

```
SELECT ExternalId, Name, Position, Status, Tip, Type
FROM EventBusSubscriber
WHERE Topic='Low_Ink__e'
```

EventBus Class

Contains methods for publishing platform events.

Namespace

System

Usage

To learn how to use platform events in Apex, see Publish Platform Events.

EventBus Methods

EventBus Methods

The following are methods for EventBus. All methods are static.

publish(event)
Publishes the given platform event. To receive published events, use triggers for the corresponding event object.

publish(events)
Publishes the given list of platform events. To receive published events, use triggers for the corresponding event object.

`publish(event)`

Publishes the given platform event. To receive published events, use triggers for the corresponding event object.

Signature

```
public static Database.SaveResult publish(SObject event)
```

Parameters

event
Type: SObject

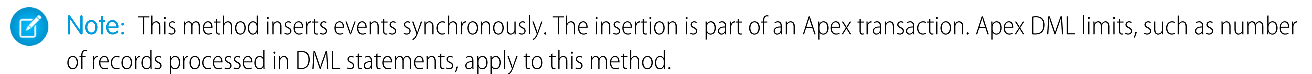
An instance of a platform event. You must define your platform event object first in your org. For example, the type of the platform event object can be `MyEvent__e`.

Return Value

Type: Database.SaveResult

The result of publishing the given event.

Usage



Note: This method inserts events synchronously. The insertion is part of an Apex transaction. Apex DML limits, such as number of records processed in DML statements, apply to this method.

`publish(events)`

Publishes the given list of platform events. To receive published events, use triggers for the corresponding event object.

Signature

```
public static List<Database.SaveResult> publish(List<SObject> events)
```

Parameters

events
Type: List<sObject>

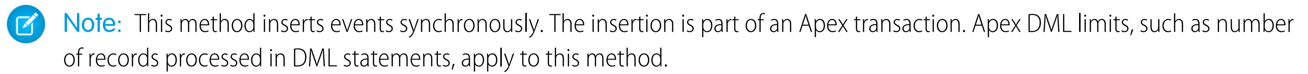
A list of platform event instances. You must define your platform event object first in your org. For example, the type of the platform event object can be `MyEvent__e`.

Return Value

Type: List<Database.SaveResult>

A list of results, each corresponding to the result of publishing one event.

Usage



Note: This method inserts events synchronously. The insertion is part of an Apex transaction. Apex DML limits, such as number of records processed in DML statements, apply to this method.

TriggerContext Class

Provides information about the trigger that's currently executing, such as how many times the trigger was refired due to the `EventBus.RetryableException`.

Namespace

EventBus

TriggerContext Properties

TriggerContext Methods

TriggerContext Properties

The following are properties for `TriggerContext`.

lastError
Read-only. The error message that the last thrown `EventBus.RetryableException` contains.

retries
Read-only. The number of times the trigger was refired due to throwing the `EventBus.RetryableException`.

lastError

Read-only. The error message that the last thrown `EventBus.RetryableException` contains.

Signature

```
public String lastError {get;}
```

Property Value

Type: String

Usage

The error message that this property returns is the message that was passed in when creating the `EventBus.RetryableException` exception, as follows.

```
throw new EventBus.RetryableException(
             'Condition is not met, so retrying the trigger again.');
```

`retries`

Read-only. The number of times the trigger was refired due to throwing the `EventBus.RetryableException`.

Signature

```
public Integer retries {get;}
```

Property Value

Type: Integer

TriggerContext Methods

The following are methods for `TriggerContext`.

currentContext()
Returns an instance of the `EventBus.TriggerContext` class containing information about the currently executing trigger.

`currentContext()`

Returns an instance of the `EventBus.TriggerContext` class containing information about the currently executing trigger.

Signature

```
public static eventbus.TriggerContext currentContext()
```

Return Value

Type: EventBus.TriggerContext

Information about the currently executing trigger.