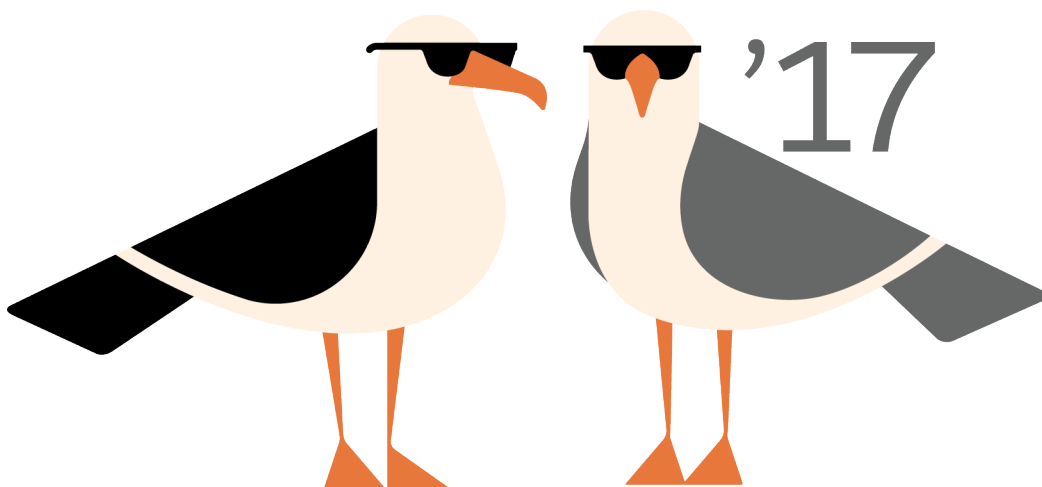

Platform Events Developer Guide

Version 40.0, Summer '17



CONTENTS

Chapter 1: Delivering Custom Notifications with Platform Events	1
Event-Driven Software Architecture	1
Enterprise Messaging Platform Events	2
Chapter 2: Defining Your Platform Event	4
Platform Event Fields	4
Migrate Platform Event Definitions with Metadata API	5
Chapter 3: Publishing Platform Events	8
Publish Event Messages Using Apex	8
Publish Event Messages Using Processes	9
Publish Event Messages Using Flows	10
Publish Event Messages Using Salesforce APIs	11
Chapter 4: Subscribing to Platform Events	13
Subscribe to Platform Event Notifications with Apex Triggers	13
Retry Event Triggers with EventBus.RetryableException	15
Subscribe to Platform Event Notifications with CometD	15
Obtain Apex Trigger Subscribers	16
Chapter 5: Platform Event Considerations and Testing	18
Platform Event Considerations	18
Apex, API, and SOQL Considerations for Platform Events	19
Test Your Platform Event Trigger in Apex	20
What Is the Difference Between the Salesforce Events?	21
Chapter 6: Reference	22
Platform Event Limits	22
EventBusSubscriber	23
EventBus Class	25
EventBus Methods	25
TriggerContext Class	26
TriggerContext Properties	27
TriggerContext Methods	28

CHAPTER 1 Delivering Custom Notifications with Platform 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.

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

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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.

Enterprise Messaging Platform Events

The Salesforce enterprise messaging platform 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 action. 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. One or more subscribers can listen to the same event and carry out actions.

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.

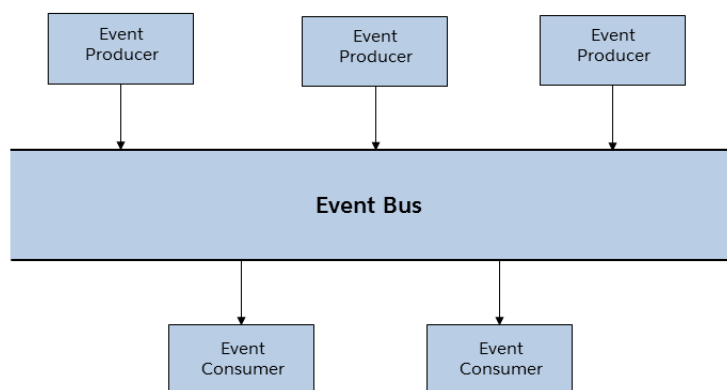
Event consumer

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. 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 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 action. 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. One or more subscribers can listen to the same event and carry out actions.

For example, a software system can send events containing information about printer ink cartridges. Subscribers can subscribe to the events to monitor printer ink levels and place orders to replace cartridges with low ink levels.

Platform events are defined in Salesforce in the same way that you define custom objects. Create a platform event definition by giving it a name and adding custom fields. The custom fields define the data that is sent in the event messages. Platform events support a subset of field types in Salesforce. See [Platform Event Fields](#). This table lists a sample definition of custom fields for a printer ink event.

Field Name	Field API Name	Field Type
Printer Model	Printer_Model__c	Text
Serial Number	Serial_Number__c	Text
Ink Percentage	Ink_Percentage__c	Number

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. When an external app through the API or a native Force.com app through Apex publishes the event message, a trigger on that event is fired. Triggers run the actions in response to the event notifications. Using the printer ink example, a software system monitoring a printer makes an API call to publish

an event when the ink is low. The printer event message contains 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.

External apps can listen to event messages by subscribing to a channel through CometD. Platform apps, such as Visualforce pages and Lightning components, can subscribe to event messages with CometD as well.

Platform Events and sObjects

A platform event is a special kind of Salesforce entity, similar in many ways to an sObject. An event message is an instance of a platform event, similar to how a record is an instance of a custom object. Unlike custom objects, you can't update or delete event records. You also can't view event records in the Salesforce user interface, and platform events don't have page layouts. When you delete a platform event definition, it's permanently deleted.

You can set read and create permissions for platform events. Grant permissions to users in profiles or in permission sets.

Platform Events and Transactions

Unlike custom objects, platform events aren't processed within database transactions in the Force.com platform. As a result, publishing platform events can't be rolled back. Note the following:

- The `allOrNoneHeader` API header is ignored when publishing platform events through the API.
- The Apex `setSavepoint()` and `rollback()` Database methods aren't supported with platform events.

When publishing platform events, DML limits and other Apex governor limits apply.

CHAPTER 2 Defining Your Platform Event

Platform events are sObjects, similar to custom objects. Define a platform event in the same way you define a custom object.

Platform Event Fields

Platform events contain standard fields. Add custom fields for your custom data.

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.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To create and edit platform event definitions:

- Customize Application

Platform Event Fields

Platform events contain standard fields. Add custom fields for your custom data.

To define a platform event in Salesforce Classic or Lightning Experience:

1. From Setup, enter *Platform Events* in the Quick Find box, then select **Platform Events**.
2. On the Platform Events page, click **New Platform Event**.
3. Complete the standard fields, and optionally add a description.
4. Click **Save**.
5. To add a field, in the Custom Fields & Relationships related list, click **New**.
6. Follow the custom field wizard to set up the field properties.



Note: In Lightning Experience, platform events aren't shown in the Object Manager's list of standard and custom objects and aren't available in Schema Builder.

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."

Field	Description
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` field identifies a platform event record and is used to get stored events that are within the retention window. Only a subscribed CometD client can retrieve past events using the `ReplayId` field. SOQL queries aren't supported. This field is among the standard fields that the system populates for each platform event. 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.

API Name Suffix for Platform Events

When you create a platform event, the system appends the `__e` suffix 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.

SEE ALSO:

[Platform Event Considerations](#)

[Apex, API, and SOQL Considerations for Platform Events](#)

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 a number field and two text fields.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
  <deploymentStatus>Deployed</deploymentStatus>
  <fields>
    <fullName>Ink_Percentage__c</fullName>
    <externalId>>false</externalId>
    <isFilteringDisabled>>false</isFilteringDisabled>
    <isNameField>>false</isNameField>
    <isSortingDisabled>>false</isSortingDisabled>
    <label>Ink Percentage</label>
    <precision>18</precision>
    <required>>false</required>
    <scale>2</scale>
    <type>Number</type>
    <unique>>false</unique>
  </fields>
  <fields>
    <fullName>Printer_Model__c</fullName>
    <externalId>>false</externalId>
    <isFilteringDisabled>>false</isFilteringDisabled>
    <isNameField>>false</isNameField>
    <isSortingDisabled>>false</isSortingDisabled>
    <label>Printer Model</label>
    <length>20</length>
    <required>>false</required>
    <type>Text</type>
    <unique>>false</unique>
  </fields>
  <fields>
    <fullName>Serial_Number__c</fullName>
    <externalId>>false</externalId>
    <isFilteringDisabled>>false</isFilteringDisabled>
    <isNameField>>false</isNameField>
    <isSortingDisabled>>false</isSortingDisabled>
    <label>Serial Number</label>
    <length>20</length>
    <required>>false</required>
    <type>Text</type>
    <unique>>false</unique>
  </fields>
  <label>Low Ink</label>
  <pluralLabel>Low Ink</pluralLabel>
</CustomObject>
```

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

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

```
<members>Low_Ink__e</members>
<name>CustomObject</name>
</types>
<version>40.0</version>
</Package>
```

To retrieve all platform events, use the wildcard character (*) for the <members> element, as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
  <types>
    <members>*</members>
    <name>CustomObject</name>
  </types>
  <version>40.0</version>
</Package>
```

To retrieve or deploy triggers associated to a platform event, use the ApexTrigger metadata type. For more information about how to use Metadata API and its types, see the [Metadata API Developer Guide](#).

CHAPTER 3 Publishing Platform Events

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.

[Publish Event Messages Using Processes](#)

Use Process Builder to publish event messages from a Force.com app as part of an automated process.

[Publish Event Messages Using Flows](#)

Use Visual Workflow to publish event messages from a Force.com app as part of some user interaction, an automated process, Apex, or workflow action.

[Publish Event Messages Using Salesforce APIs](#)

External apps use an API to publish platform event messages.

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.');
```

```

err.getStatusCode() +
' - ' +
err.getMessage());
    }
}

```

The event insertion occurs non-transactionally. As a result, you can't roll back published events. Because event publishing is equivalent to a DML insert operation, DML limits and other Apex governor limits apply.

Publish Event Messages Using Processes

Use Process Builder to publish event messages from a Force.com app as part of an automated process.

To publish event messages, add a Create a Record action to the appropriate process. Where you'd usually pick an object to create, select the platform event.

For example, here's how to configure a Create a Record action that publishes a Low Ink event message. This example assumes that the Low Ink platform event is defined in your org and that the event has these custom fields.

- Printer Model (Text)
 - Serial Number (Text)
 - Ink Percentage (Number)
1. For Record Type, enter *Low* and select **Low Ink**.
 2. Set the field values.

Field	Type	Value
Printer Model	String	XZO-5
Serial Number	String	12345
Ink Percentage	Number	0.2

Action Type *

Create a Record

Action Name * ⓘ

Publish Low Ink Event

Record Type *

Low Ink

Set Field Values

Field *	Type *	Value *
Printer Model	String	XZO-5
Serial Number	String	12345
Ink Percentage	Number	0.2

+ Add Row

3. Save the action and activate the process.

SEE ALSO:

[Salesforce Help: Lightning Process Builder](#)

Publish Event Messages Using Flows

Use Visual Workflow to publish event messages from a Force.com app as part of some user interaction, an automated process, Apex, or workflow action.

To publish event messages, add a Record Create or a Fast Create element to the appropriate flow. Where you'd usually pick an object to create, select the platform event.

For example, here's how to configure a Record Create element that publishes a Low Ink event message. This example assumes that the Low Ink platform event is defined in your org and that the event has these custom fields.

- Printer Model (Text)
 - Serial Number (Text)
 - Ink Percentage (Number)
1. For Create, enter `Low` and select **Low_Ink__e**.
 2. Add these fields.

Field	Value
Printer Model	XZO-5
Serial Number	12345
Ink Percentage	0.2

Record Create

Select the type of record you want to create, then insert flow values into its fields.

General Settings

Name *

Publish Low Ink Event

Unique Name *

Publish_Low_Ink_Event

Add Description

Assignments

Create *

Low_Ink__e

with the following field values:

Field	Value
Printer_Model__c	XZO-5
Serial_Number__c	12345
Ink_Percentage__c	0.2

Add Row

OK

Cancel

3. Save and activate the flow.

SEE ALSO:

[Visual Workflow Guide](#)

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/v40.0/subjects/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:subject.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

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.

[Subscribe to Platform Event Notifications with CometD](#)

Use CometD to subscribe to platform events in an external client. A Java library and app, EMP Connector, implements all the details of connecting to CometD and listening on a channel. Use EMP Connector to subscribe to platform events easily. EMP Connector hides the complexity of subscribing to events.

[Obtain Apex Trigger Subscribers](#)

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

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.

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 the `Printer_Model__c` 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;
}

```

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. The order of events is preserved within each batch. The events in a batch can originate from one or more publishers.

Unlike triggers on standard or custom objects, triggers on platform events don't execute in the same Apex transaction as the one that published the event. The trigger runs asynchronously in its own process under the Automated Process entity. As a result, there might be a delay between when an event is published and when the trigger processes the event. Also, debug logs corresponding to the trigger execution are created by Automated Process. System fields, such as `CreatedById` and `LastModifiedById`, reference the Automated Process entity.



Note: If you create a Salesforce record with an `OwnerId` field in the trigger, such as a case or opportunity, 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](#).

Event triggers have many of the same limitations of custom and standard object triggers. For example, you can't make Apex callouts from triggers with some exceptions. For more information, see [Implementation Considerations for Triggers](#) in the *Apex Developer Guide*. Also, the trigger is subject to Apex governor limits.

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 entity in Setup.

1. From Setup, enter *Debug Logs* in the Quick Find box, then click **Debug Logs**.
2. Click **New**.
3. For Traced Entity Type, select **Automated Process**.
4. Select the time period to collect logs and the debug level.
5. Click **Save**.

To collect logs in the Debug Logs page for your Apex code that publishes the events, add another trace flag entry for your user.

For more information, see [Set Up Debug Logging](#) in the *Salesforce Help*.

[Retry Event Triggers with EventBus.RetryableException](#)

Get another chance to process event notifications. Retrying a trigger is helpful when a transient error occurs or when waiting for a condition to change. Retry a trigger if the error or condition is external to the event records and is likely to go away later.

SEE ALSO:

[Apex, API, and SOQL Considerations for Platform Events](#)

Retry Event Triggers with `EventBus.RetryableException`

Get another chance to process event notifications. Retrying a trigger is helpful when a transient error occurs or when waiting for a condition to change. Retry a trigger if the error or condition is external to the event records and is likely to go away later.

An example of a transient condition: 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 retry the event trigger, throw `EventBus.RetryableException`. The event is resent after a small delay. The delay increases in subsequent retries. 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.

You can run a trigger up to 10 times when it is retried (the initial run and 9 retries). After the trigger is retried 9 times, it moves to the error state and stops processing new events. To resume event processing, fix the trigger and save it. Events sent after the trigger moves to the error state and before it returns to the running state are not resent to the trigger. We recommend setting a limit to retry the trigger to less than 9 times. Use the `EventBus.TriggerContext.currentContext().retries` property to check how many times the trigger has been retried.

This example is a skeletal trigger that gives you an idea of how to throw `EventBus.RetryableException` and limit the number of retries. 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 {
    // Ensure we don't retry the trigger more than 4 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.');
```

Subscribe to Platform Event Notifications with CometD

Use CometD to subscribe to platform events in an external client. A Java library and app, EMP Connector, implements all the details of connecting to CometD and listening on a channel. Use EMP Connector to subscribe to platform events easily. EMP Connector hides the complexity of subscribing to events.

Salesforce sends platform events to CometD clients 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
```

Use this CometD endpoint with the API version appended to it.

```
/cometd/40.0
```

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

```
{
  "data": {
    "schema": "dffQ2QLzDNHqwB8_sHMxdA",
    "payload": {
      "CreatedDate": "2017-04-09T18:31:40Z",
      "CreatedById": "005D0000001cSZs",
      "Printer_Model__c": "XZO-5",
      "Serial_Number__c": "12345",
      "Ink_Percentage__c": 0.2
    },
    "event": {
      "replayId": 2
    }
  },
  "channel": "/event/Low_Ink__e"
}
```

The `schema` field in the event message contains the ID of the platform event schema. The schema is versioned—when the schema changes, the schema ID changes as well.

To determine if the schema of an event has changed, retrieve the schema through REST API. Use the schema ID by performing a GET request to this REST API resource: `/vXX.X/event/eventSchema/Schema_ID`. Alternatively, you can retrieve the event schema by supplying the event name to this endpoint: `/vXX.X/subjects/Platform_Event_Name__e/eventSchema`. For more information, see:

- [Platform Event Schema by Schema ID](#) in the *Force.com REST API Developer Guide*
- [Platform Event Schema by Event Name](#) in the *Force.com REST API Developer Guide*

You can use EMP Connector to receive delivered events. The connector subscribes to streaming events and platform events—only the supplied 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.

SEE ALSO:

[Apex, API, and SOQL Considerations for Platform Events](#)

Obtain Apex Trigger 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.

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 active triggers that are subscribed to platform events. The list shows the replay ID of the event that the system last processed and the event last published. Knowing which replay ID was last processed is useful when there is a gap in the events published and processed. For example, if a trigger contains complex logic that causes a delay in processing large batches of incoming events.

Also, the Subscriptions list shows the state of each subscribed trigger. Trigger states can be one of the following.

- **Running**—The trigger is actively listening to events.
- **Idle**—The trigger hasn't received events for some time and is not actively listening to events. When new events are sent, the trigger receives the new events after a short delay and switches to the **Running** state.
- **Error**—The trigger has been disconnected and stopped receiving published events. A trigger reaches this state when it exceeds the number of maximum retries with the `EventBus.RetryableException`. Trigger assertion failures and unhandled exceptions don't cause the Error state. To resume trigger execution, fix the trigger code and save it.
- **Suspended**—The trigger is disconnected and can't receive events due to lack of permissions.
- **Expired**—The trigger's connection expired. In rare cases, subscriptions can expire if they're inactive for an extended period of time.

Alternatively, you can obtain the same subscriber information by querying the `EventBusSubscriber` object. See [EventBusSubscriber](#).

CHAPTER 5 Platform Event Considerations and Testing

Learn about special behaviors related to defining, publishing, and subscribing to platform events. Learn how to test platform events. And get an overview of the various events that Salesforce offers.

Platform Event Considerations

Take note of the considerations when defining platform events.

Apex, API, and SOQL Considerations for Platform Events

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

Test Your Platform Event Trigger in Apex

Ensure that your platform event trigger is working properly by adding an Apex test. Before you can package or deploy any Apex code (including triggers) to production, your Apex code must have tests. To publish platform events in an Apex test, enclose the publish statements within `Test.startTest()` and `Test.stopTest()` statements.

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.

Platform Event Considerations

Take note of the considerations when defining platform events.

Field-Level Security

All platform event fields are read-only by default, and you can't restrict access to a particular field. Because platform events aren't viewable in the Salesforce user interface and aren't editable, field-level security permissions don't apply.

Enforcement of Field Attributes

Platform event records are validated to ensure that the attributes of their custom fields are enforced. Field attributes include the Required and Default attributes, the precision of number fields, and the maximum length of text fields.

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.

Platform Events in Package Uninstall

When uninstalling a package with the option **Save a copy of this package's data for 48 hours after uninstall** enabled, platform events aren't exported.

No Support in Professional and Group Editions

Platform events aren't supported in Professional and Group Edition orgs, even if a package containing platform event objects is installed in the orgs.

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 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

In platform event triggers, if you create a Salesforce record that contains an `ownerId` field, set the `ownerId` field 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(
    OwnerId = customerOrder.createdById,
    AccountId = acc.Id,
    StageName = 'Qualification',
    Name = 'A ' + customerOrder.Product_Name__c + ' opportunity for ' + acc.name,
    CloseDate = Date.today().addDays(7));
```

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.

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.

- [Streaming API Developer Guide: Message Durability](#)
- [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



Note: In rare occasions, some maintenance activities on the Salesforce application servers wipe retained events. For example, maintenance activities, such as disaster recovery of servers after unexpected outages, a site switch of a Salesforce instance, or an org migration to a new data center, can wipe retained events. To prevent losing events, receive only the latest events by subscribing to the tip of the queue (-1 replay option).

Filtered Subscriptions

Filtered subscriptions in Streaming API aren't supported for platform events.

No SOQL Support

You can't query event notifications using SOQL.

SEE ALSO:

[Platform Event Limits](#)

Test Your Platform Event Trigger in Apex

Ensure that your platform event trigger is working properly by adding an Apex test. Before you can package or deploy any Apex code (including triggers) to production, your Apex code must have tests. To publish platform events in an Apex test, enclose the publish statements within `Test.startTest()` and `Test.stopTest()` statements.

Call the `publish` method within the `Test.startTest()` and `Test.stopTest()` statements. In test context, the `publish` method enqueues the publish operation. The `Test.stopTest()` statement causes the event publishing to be carried out. Include your validations after the `Test.stopTest()` statement.

```
// Create test events
Test.startTest();
// Publish test events
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 and the associated trigger to be defined in the org.

```
@isTest
public class EventTest {
    @isTest static void test1() {

        // Create a test event instance
        Low_Ink__e inkEvent = new Low_Ink__e(Printer_Model__c='MN-123',
                                              Serial_Number__c='10013',
                                              Ink_Percentage__c=0.15);

        Test.startTest();

        // Publish test event
        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());
}

}
```

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 6 Reference

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

[Platform Event Limits](#)

Limits are enforced for publishing platform events, event delivery in CometD clients, and platform event definitions.

[EventBusSubscriber](#)

Represents a trigger that is subscribed to a platform event.

[EventBus Class](#)

Contains methods for publishing platform events.

[TriggerContext Class](#)

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

Platform Event Limits

Limits are enforced for publishing platform events, event delivery in CometD clients, and platform event definitions.

Description	Performance and Unlimited Editions	Enterprise Edition	All other supported editions
Maximum number of events published per hour (Events can be published using Apex or APIs.)	100,000	100,000	1,000
Maximum event message size	5 MB		
Maximum number of events delivered to CometD clients within a 24-hour period ¹	50,000	25,000	10,000
Maximum number of platform event definitions that can be created in an org	100	50	5

¹To request a higher number of events delivered to CometD clients, contact Salesforce to purchase an add-on license. The add-on license increases your daily limit of delivered events by 100,000 more events. For example, for Unlimited Edition, the add-on license increases the daily limit of delivered events from 50,000 to 150,000 events. You can purchase multiple add-ons to meet your event requirements for CometD clients. To avoid deployment problems and degradation in service, we recommend that the number of events delivered to CometD clients not exceed 5 million per day. If you require additional external events, contact your Salesforce representative to understand how the product can scale to meet your needs.

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 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:

Field	Details
	<ul style="list-style-type: none"> • Running—The trigger is actively listening to events. • Idle—The trigger hasn't received events for some time and is not actively listening to events. When new events are sent, the trigger receives the new events after a short delay and switches to the Running state. • Error—The trigger has been disconnected and stopped receiving published events. A trigger reaches this state when it exceeds the number of maximum retries with the <code>EventBus.RetryableException</code>. Trigger assertion failures and unhandled exceptions don't cause the Error state. To resume trigger execution, fix the trigger code and save it. • Suspended—The trigger is disconnected and can't receive events due to lack of permissions. • Expired—The trigger's connection expired. In rare cases, subscriptions can expire if they're inactive for an extended period of time.
Tip	<p>Type int</p> <p>Properties Filter, Group, Nillable, Sort</p> <p>Description The replay ID of the last published event.</p>
Topic	<p>Type string</p> <p>Properties Filter, Group, Nillable, Sort</p> <p>Description The name of the subscription channel that corresponds to a platform event. The topic name is the event name appended with <code>__e</code>, such as <code>MyEvent__e</code>. The topic is the channel that the subscriber is subscribed to.</p>
Type	<p>Type string</p> <p>Properties Filter, Group, Nillable, Sort</p> <p>Description The subscriber type. Can be one of the following values:</p> <ul style="list-style-type: none"> • 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

[EventBus Methods](#)

SEE ALSO:

[Platform Events Developer Guide: Publishing Platform Events](#)

EventBus Methods

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

[publish\(event\)](#)

Publishes the given platform event.

[publish\(events\)](#)

Publishes the given list of platform events.

publish(event)

Publishes the given platform event.

Signature

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

Parameters

event

Type: `SObject`

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

Return Value

Type: `Database.SaveResult`

The result of publishing the given event. `Database.SaveResult` contains information about whether the operation was successful and the errors encountered. If the `isSuccess()` method returns `true`, the event was inserted.

Usage



Note:

- The event insertion occurs non-transactionally. As a result, you can't roll back published events.
- Apex governor limits apply, including DML limits. Each method execution is counted as one DML statement.

publish(events)

Publishes the given list of platform events.

Signature

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

Parameters

events

Type: `List<SObject>`

A list of platform event instances. For example, a list of `MyEvent__e` objects. You must first define your platform event object in your org.

Return Value

Type: `List<Database.SaveResult>`

A list of results, each corresponding to the result of publishing one event. For each event, `Database.SaveResult` contains information about whether the operation was successful and the errors encountered. If the `isSuccess()` method returns `true`, the event was inserted.

Usage



Note:

- The event insertion occurs non-transactionally. As a result, you can't roll back published events.
- Apex governor limits apply, including DML limits. Each method execution is counted as one DML statement.

TriggerContext Class

Provides information about the trigger that's currently executing, such as how many times the trigger was retried 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 retried 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 retried 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.