# Standard for receiving audit-messages from BankID self-service ( BASS )

This integration guide is designed to help you, as a client receiving audit events from our service, set up and manage the integration. The standards used for delivering events are CloudEvents delivered through Webhooks over HTTPS.

When the user resets their BankID password, messages are sent in two rounds:

1. One is sent before we send a command to the RA, signaling the intent to perform a password change. At this point, the user can not abort.
2. One is sent after we have sent the command to the RA and received a response, with OK or Failed status.

## Table of Contents


**1. Introduction**

1.1 Audit events in the future

**2. Integration Steps**

2.1. Send us required information
2.2. Validate subscription
2.3. Receive audit events
2.4 Example Cloudevents subscriber

**3. Handling Cloudevents**

3.1 Types
3.2 Data fields
3.3 Handle incorrect messages

**4. Advanced Configuration**

4.1. Webhook URL
4.2. Authentication

**5. Troubleshooting**

**6. Further documentatin**


## 1. Introduction

From the Cloudevents documentation;

> *CloudEvents is a specification for describing event data in common formats to provide interoperability across services, platforms and systems.*

By utilizing Cloudevents delivered to a Webhook over HTTPS, we adhere to standard protocols decoupled from specific libraries or SDKs. In this guide, we aim to provide you with the necessary information to set up a receiver capable of receiving audit events from our systems.

### 1.1 Audit events in the future

While the current integration only send audit events related to password reset, this will likely be expanded in the future with new messages. Some example use-cases for sending out audit events are;

- App activation as OTP
- Automatic block of BankID certificate based on anti-fraud signals
- Biometric activation

## 2. Integration Steps

### 2.1. Send us required information

The required information is the following:

- **Webhook url**: This is the full URL you will receive events on. It has to be publicly accessible from the internet, in all environments.
- **Authentication**: One of
    - **Static token**: A static token sent in the Authorization header
    - **AppId and TenantId**: Application ID for our application registration in your Azure AD tenant and Tenant ID for your Azure tenant.

For further information regarding authentication, see separate bulletpoint bellow.

## 2.2. Validate subscription

Once we attempt to create the subscription, a subscription validation request will be sent to the provided URL by a **HTTP Options** call. A header, **WebHook-Request-Origin**, will be included in this request. It contains a DNS expression identifying the system sending the events. At the time of writing, this will be "**eventgrid.azure.net**"

To accept the subscription, reply with HTTP Status 200 and the following two headers:

- **WebHook-Allowed-Origin**: One of two possible values; the same value as was sent in the WebHook-Request-Origin header or an asterix("*")
- **WebHook-Allowed-Rate**: An asterix("*") to indiciate no rate limitation, or a positive integer number to indicate requests per minute.

There are more headers sent in the request, both standard http-headers and application-specific headers, but the headers mentioned here are the ones you have to strictly handle to validate the subscription

## 2.3. Receive audit events

Events will be delivered in two rounds – one right before the reissue command is sent to FOI, and one right after indicating the status. These are sent asynchronously, meaning that the command might have been executed before you receive the events. If the delivery fails, a retry with exponential backoff will be retried on a best-effort basis. If the delivery still fails after 24 hours, the event is sent to our dead-letter storage for manual processing and debugging.

Response code for a successfull delivery **must** be one of **200**, **201**, **202**, **203** or **204**. A response code of **400**, **401**, **403**, **404** and **413** will stop retries and put the event in our dead-letter event storage. Other failure codes will be retried as described above. If repeated failures occurs during a short time, all deliveries to the endpoint may be suspended for up to several hours.

## 2.4. Example Cloudevents subscriber

An example implementation of a Cloudevents subscriber can be found here; https://github.com/BankIDNorge/example-cloudevent-subscriber

It is written in Typescript for Azure Functions.

# 3. Handling Cloudevents

A reworked example request from the Cloudevents documentation can be seen here:

```
POST /your-subscription HTTP/1.1
Host: webhook.example.com
Content-Type: application/cloudevents+json; charset=utf-8
Content-Length: nnnn
Authorization: a-token

{
    "specversion" : "1.0",
    "type" : "com.example.someevent",
    ... further attributes omitted ...
    "data" : {
        ... application data ...
    }
}
```

Two main things to note about the payload are the type and data fields.

## 3.1 Types

Type will have one of two values:

- **no.bankid.bass.audit.reissue.init.v1** Used for events sent right *before* the reissue command is sent. This signals the intent that the user will now perform a password reset, and the user can no longer abort.
- **no.bankid.bass.audit.reissue.completed.v1** Used for events sent right *after* the reissue command is sent. This signals that the command has been sent, and will specify if the command was successful or failed.

## 3.2 Data fields

Data will be a json with the following specification;

| Parameter | Type | Possible values | Sample data | Description | Required |
|---|---|---|---|---|---|
| **sessionid** | String | | 7468bdd3-274b-4e2f-b7bb-65dad59ce8a9 | SessionID of the transaction. Is only valid for a single password reset. | Yes |
| **authentication** | String | BIM | BIM | Authentication method used to identify the user | Yes |
| **action** | String | REISSUE | REISSUE | Actual operation | Yes |

| | | | | | |
|---|---|---|---|---|---|
| **orderid** | String | | 1012-1667319077298 | OrderID of the BankID | Yes |
| **time** | String | | 2022-10-26T14:15:51.978Z | Execution timestamp(ISO8601) of the reissue command. Not set for events with status=BEGIN | No |
| **status** | String | BEGIN, SUCCESS, FAILURE | SUCCESS | Status of the transaction. | Yes |
| **additionalInfo** | String | | | Details information if applicable to action, like error message in case of failure | No |

### 3.3 Handle incorrect messages

While the messages are defined, typos and errors may occur on either end. As such, you need to have proper handling of errors related to the messages - such as, but not limited to, unknown type, unknown data fields, missing data fields, unknown data values etc.

If you discover incorrect messages, contact us for troubleshooting. Please include the full message and time of delivery.

## 4. Security

### 4.1. Webhook URL

The following requirements for the webhook URL are absolute:

- It must be publicly accessible
- It must be available via HTTPS with a valid certificate
- It must accept Options and Post calls

### 4.2. Authentication

To prevent anyone from calling your subscription webhooks, there needs to be a layer of authentication. We support the following three options:

1. Static token in the authorization header
2. Azure AD App

#### Authorization header

For the static options, we need to agree on a routine for rolling the secret. You will also need to accept both the old and the new token, at least for a short periode of time, while we update our system with the new token. The format will be `authorization: api-key <static-token>`.

#### Azure AD App

For Azure AD App, you need to perform the following steps:

1. Create an **Azure AD app for your Webhook**.
2. Create a service principal for **Microsoft.EventGrid** if it doesn't already exist.
3. Create a role named **AzureEventGridSecureWebhookSubscriber** in the **Azure AD app for your Webhook**.
4. Create a service principal for our **event subscription writer app** if it doesn't already exist.
5. Add service principal of event subscription writer Azure AD app to the AzureEventGridSecureWebhookSubscriber role
6. Add service principal of Microsoft.EventGrid to the AzureEventGridSecureWebhookSubscriber role as well

Link to source for these steps: Secure WebHook delivery with Azure AD Application in Azure Event Grid
Note that the link also contains a Powershell script which can perform the setup. For step 4, and the `eventSubscriptionWriterAppId` variable, you'll need the relevant applicationID from us, which will be provided on request.

We have one app for test and one for production, so some or all of the steps will have to be duplicated depending on your setup. If you decide on creating one Azure AD App for the webhook in each environment, all but the second step will have to be duplicated. If you go for just one Azure AD App, only step 4 and 5 will have to be duplicated. appId for the two apps will be provided to you on request.

Once you have finished setting up the service principals, roles and Azure AD Apps, send us the following info:

- Your Azure tenant ID
- appId of the Azure AD Apps that was created

## 5. Troubleshooting

### Validation of subscription fails

- Check that firewalls are open according to 4.1.
- Check that you accept the correct Content-Type("application/cloudevents+json; charset=utf-8")

## 6 Further documentation

- Azure Event Grid: https://learn.microsoft.com/en-us/azure/event-grid/
- Endpoint validation with Azure Event Grid: https://learn.microsoft.com/en-us/azure/event-grid/webhook-event-delivery
- Event Grid CloudEvents Schema: https://learn.microsoft.com/en-us/azure/event-grid/cloud-event-schema
- Webhook delivery with Azure AD: https://learn.microsoft.com/en-us/azure/event-grid/secure-webhook-delivery