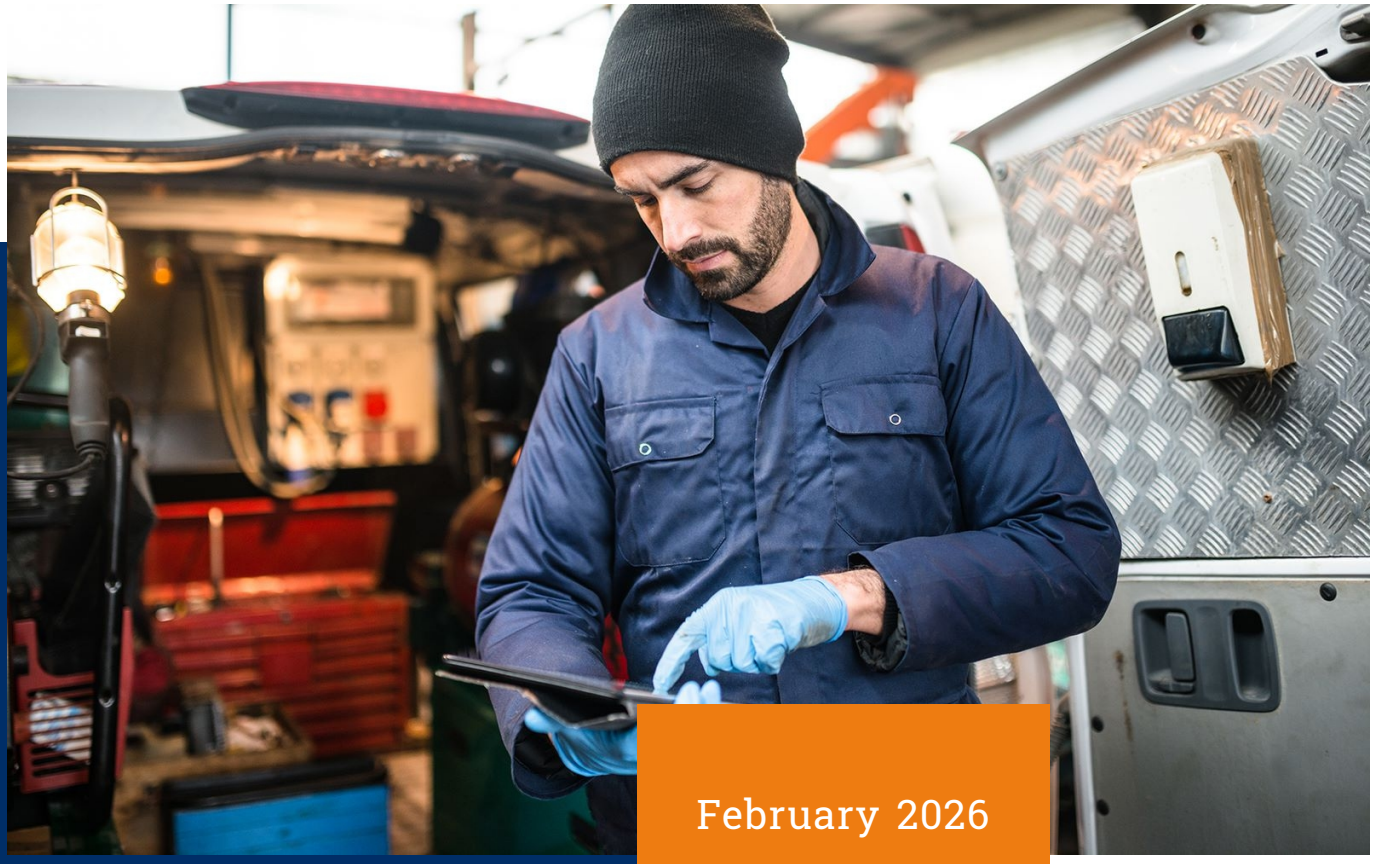
ORTEC for Field Service

Webhooks Subscriptions and Event Processing



February 2026

PUBLIC

ORTEC

Published: February 24, 2026

Version: 2026.1703218

Table of Contents

1 Introduction

When a resource (entity) is created or updated in ORTEC for Field Service (ORTEC FS), an event is generated. Webhooks act as a notification mechanism for these events, enabling authorized third-party applications (such as an ERP or a CRM solution) to be informed about changes in your data. This provides an efficient way of ensuring that the necessary actions can be triggered to keep your data consistent and up-to-date across multiple applications.

1.1 What's in this document?

This document describes guidelines that should be followed by client applications when implementing webhooks. Following are the topics covered:

- Prerequisites
- How to create event subscriptions
- How to handle webhook calls
- Retry policy for failed requests

The contents of this document are most relevant to API developers who are implementing webhooks in their application.

The features described in this document apply to ORTEC for Field Service. If you have any questions, please contact your ORTEC representative.

2 Prerequisites

This chapter describes the prerequisites that must be in place before you can create webhook subscriptions and start handling webhook calls.

2.1 Authorization

ORTEC provides API authorization credentials to create webhook subscriptions and make calls to the webhook endpoints. These credentials are bound to a specific set of permissions. The same set of permissions is considered for sending event notifications. Every webhook subscription only gets event notifications for the data it's permitted to access.

2.2 Callback URL

You must set up a webhook endpoint (callback URL). ORTEC FS requires that this endpoint meets the following conditions:

- Accept and respond to POST requests.
- Support secure communication (HTTPS).
- Return success or failure with response status codes:
  - in the 2xx range for successfully received requests.
  - in the 4xx range for invalid requests.
  - in the 5xx range for transient failures.
- Parse JSON payload.
- Be able to receive and process HTTP headers included in the webhook request, such as the X-Signature header, an Authorization header, or configured custom headers.
- Support idempotency: ORTEC FS retries to send the same request multiple times in case of errors or time-outs.

2.3 Callback security requirements

Webhook requests include authentication information based on the oauth1, oauth2, or headers configuration defined when creating the webhook subscription. Depending on the configuration of the webhook subscription, the following authentication mechanisms can be used:

- **JSON Web Signature (JWS)**
  Each request contains an X-Signature header that can be validated using ORTEC public keys.
- **OAuth 1.0 Authentication**
  Requests include an OAuth 1.0 Authorization header generated using the configured OAuth1 parameters.
- **OAuth 2.0 Authentication**
  Requests include OAuth 2.0 authentication data. Processing of this information is the responsibility of the receiving system.
- **Static HTTP Headers**
  Custom headers configured in the webhook subscription are included in each request. Only the authorization header and headers whose names start with x- are supported.

The selected authentication method is defined during webhook subscription setup.

3 Creating webhook subscriptions

ORTEC FS needs to know where to send event notifications. For this purpose, you must create a separate webhook subscription for each event type. For a complete overview of supported event types, refer to the Webhooks API specification file.

You can create and maintain webhook subscriptions using the /webhooks endpoint. Subscribing to an event type (via the POST method) is a one-time action. The payload of the POST request must contain:

- The event type to which you want to subscribe.
- The callback URL where ORTEC FS needs to send the event notification. The URL must be syntactically correct.

> Our APIs enforce the HTTPS protocol for secure communication.

Example

```
{
    "event": "SiteCreated",
    "url": "https://example.hostname.com/api/v1/receiveEvent"
}
```

3.1 Additional examples

The webhook subscription request can include optional authentication or header configuration. These settings determine which HTTP headers ORTEC FS includes when sending webhook callbacks. The following examples show common configurations.

Optional HTTP headers can be configured for webhook callbacks.

> Only the authorization header and headers whose names start with x- can be configured.

Example

Subscription with OAuth 1.0 authentication:

```
{
    "event": "ShiftPlanningChanged",
    "url": "https://whatever.hostname.com/api/v1/receiveEvent",
    "oauth1":
        {
            "consumerKey": "0685bd9184jfhq22",
            "consumerSecret": "kd94hf93k423kf44",
            "accessToken": "ad180jjd733klru7",
            "tokenSecret": "pfkkdhi9sl3r4s00",
            "signatureMethod": "HMAC-SHA256",
            "realm": "example"
        }
}
```

Example

Subscription with OAuth 2.0 authentication:

```
{
    "event": "ShiftPlanningChanged",
    "url": "https://whatever.hostname.com/api/v1/receiveEvent",
    "oauth2":
        {
            "tokenUrl": "https://example.com/oauth2/token",
            "parameters":
                {
                    "client_id": "0685bd9184jfhq22",
                    "client_secret": "pfkkdhi9sl3r4s00",
                    "grant_type": "client_credentials"
                }
        }
}
```

Example

Subscription with custom authorization header:

```
{
    "event": "ShiftPlanningChanged",
    "url": "https://whatever.hostname.com/api/v1/receiveEvent",
    "headers":
        {
            "authorization": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="
        }
}
```

Example

Subscription with custom non-authorization header:

```
{
    "event": "ShiftPlanningChanged",
    "url": "https://whatever.hostname.com/api/v1/receiveEvent",
    "headers":
        {
            "x-api-key": "example-api-key-value"
        }
}
```

4 Handling a webhook call

Webhooks act as a notification mechanism for events through HTTP callbacks. The guiding principle behind this is the 'Thin Payloads and API Retrieval' security concept.

ORTEC FS sends a POST message to your pre-configured URL that contains limited, non-sensitive information about the event. To retrieve the latest representation of the entity associated with the event, your application is expected to make a subsequent API call.

When subscribed to an event type, you will receive notifications on the callback URL specified in the webhook subscription. These notifications link to specific endpoints providing additional details. For example, if you have subscribed to the event type, "OrderCreated", the link to the endpoint provides details on the new order.

The endpoint version corresponds to the webhook subscription version. If a webhook subscription is created on version 1, the notification links to a version 1 endpoint. To update a webhook subscription version, you can do a GET webhooksubscription on the current version, and use the response to make a PUT request for that webhook subscription on the new version.

The main steps for handling a webhook call are described in the following sections.

4.1 Receive event notification

When the event type to which you subscribed occurs, ORTEC FS sends a POST request to the callback URL specified in your webhook subscription. The payload of this request indicates the event type and the entity associated with the event, as shown in the following example:

Example


```
{
    "id": "6708f18e-e65e-4277-a4c5-dcced5a6d077",
    "created": "2020-07-08T14:52:42.810+00:00",
    "name": "SiteCreated",
    "department": {
        "id": "7d1adf0d-806c-4dc9-81e8-8ee191103daf",
        "_links": [
            {
                "rel": "self",
                "href": "https://example.ortecapps.com/api/v1.0/sites/7d1adf0d-806c-4dc9-81e8-
8ee191103daf",
                "method": "GET"
            }
        ]
    }
}
```

4.2 Validate event notification

Each webhook call includes the X-Signature header, and optionally other authentication headers based on the oauth1, oauth2, or headers configuration of the webhook subscription. The validation method depends on how the webhook subscription has been configured.

Webhook subscriptions support multiple authentication models. Implementations must handle this information according to their integration design.

4.2.1 Validation methods

JSON Web Signature (JWS)

The request contains an X-Signature header. Validate the signature using the procedure described below.

Each ORTEC FS webhook call (POST request) contains a header named X-Signature. The value of this header is a JSON Web Signature (JWS). The client can use the JWS to validate that the event notification originated from ORTEC FS. The following is an example of a POST request header (ensure that you are using the correct solution/product name):

Example

```
POST /registered-callback-url HTTP/1.1
Content-Type: application/json
User-Agent: ORTEC for Home Delivery Suite
X-Signature:
eyJhbGciOiJSUzI1NiIsImtpZCI6IjQwZmE4YTVjLTRjYWEtNDRjYS1iMzZkLTczMWE4YmQ0NGU4NSJ9.eyJpZCI6Ijk3MDhmMThlL
WU2NWUtNDc3Ny1hNGM1LWRjY2VkNmE2ZDA3NyIsImNyZWF0ZWQiOiIyMDIwLTA3LTA4VDE0OjUyOjQyLjgxMCswMDowMCIsIm5hbWU
iOiJEZXBhcnRtZW50Q3JlYXRlZCIsImRlcGFydG1lbnQiOnsiaWQiOiI2ZDFhZGYwZC04MDZjLTRkYzktODFlOC00ZWUxOTExMDNkY
WYiLCJfbGlua3MiOlt7InJlbCI6InNlbGYiLCJocmVmIjoiaHR0cHM6Ly9maWVsZHNlcnZpY2UtZXhhbXBsZS5vcnRlY2FwcHMuY29
tL2FwaS92MS4wL2RlcGFydG1lbnRzLzZkMWFkZjBkLTgwNmMtNGRjOS04MWU4LTRlZTE5MTEwM2RhZiIsIm1ldGhvZCI6IkdFVCJ9X
X19.examplesignaturethiswilnotpassvalidationxxxvzR_LXGLYArWFPp605EkKd_
j5h7kr3YgEduZJF2G8TN5Kuw8RhF8eaKHXRowMclu1YwtVBAVz1jT3u9ej0s9VXBtSSHLtV1ei4SmLIowukHix0tmEK5YrClshn54W
SWNY
```

The procedure to decode and verify the JWS is described as follows:

If any validation step fails, the webhook call must be rejected.

1. Validate the JWS structure:
   A JWS contains 3 dot-separated sections: <JWSHeader>.<JWSPayload>.<JWSSignature>

2. Validate the signature:

   a. Decode and parse JWSHeader, JWSPayload and JWSSignature:
   JWSHeader and JWSPayload are base64url-encoded JSON objects. JWSHeader contains information relevant for the validation of the signature, while JWSPayload should be identical to the payload of the webhook request. An example of a decoded JWSHeader is shown as follows:

Example

```
{
    "alg": "RS256",
    "kid": "abc12345-123a-4567-def8-8901234abcde"
}
```

Base64url-encoding has some minor differences compared to base64. You can find a C# example implementing a decoding function in RFC 7515 – Appendix C (https://tools.ietf.org/html/rfc7515#appendix-C).

b. Process JWSHeader as follows:
   - Verify that alg property is set to RS256. Don't accept the value none as it leads to security risks (such as allowing an attacker to forge the signature).
   - Extract the value of the kid property. This is the identifier of the key that should be used to verify the signature.

c. Retrieve public keys using the /webhook-keys endpoint: S
The response payload represents a JSON Web Keys Set (JWKS) – a set of keys containing the public key which allows verification of JWSSignature. The public key is provided in the JSON Web Key (JWK) format (https://tools.ietf.org/html/rfc7517). An example of a JWK is shown as follows:

Example

```
{
    "alg": "RS256",
    "kty": "RSA",
    "use": "sig",
    "n": "xyz_ae25c9vg1uGYoWlWlJ9mCk0mLSZJpkJuVbXB_
bUHy7lFF7MGRhPnBwNzI0N6xpTFVL1kd8gPs66HQO33gFBPeabS0iVqHU09UJs_BGL9Q0-
MBFpqt4UYmn7IOuAqrvcWKOAwWsMB6baa7D2DrZPNUTcJBdlSYa-F9r3_xyz",
    "e": "AQAB",
    "kid": "abc12345-123a-4567-def8-8901234abcde"
}
```

You should cache the key returned by this endpoint so that you don't have to retrieve it each time a webhook message needs to be verified.

Multiple keys might be available. All the keys should be cached and the proper key should be retrieved from this set by its ID.

> For security reasons, ORTEC performs key rotation. Make sure to refresh the cache every 24 hours and retrieve the new key by calling the /webhook-keys endpoint.
>
> Webhook keys are regenerated every 28 days. During a 48 hours period keys may overlap. By caching keys on the client side every 24 hours (by getting them from /webhook-keys), clients can ensure they have valid keys all the time.
>
> Usually clients have 1 or 2 valid keys. However, in certain scenarios, more (valid) keys are generated.

d. Compare the local JWS key ID to the public key ID:
To ensure you're using the correct key, verify that the key ID extracted from JWSHeader is identical to the key ID in the JWK.

e. Verify the signature:
The signature sent in the POST request header is an RSA-based JWS. This means that you have to check if the substring made up of JWSHeader and JWSPayload matches JWSSignature using the RSA SHA256 algorithm and the public keys retrieved in step 2c. For additional details, refer to RFC 7515 Appendix A.2 (https://tools.ietf.org/html/rfc7515#appendix-A.2).

Depending on the library you're using for signature verification, you may need to:

- Compute the hash of the decoded JWSHeader and JWSPayload. JWSSignature is in fact a hashed and encrypted combination of both.
- Convert the JWK to the Privacy Enhanced Mail (PEM) format.

3. Validate the event:
To check the integrity of the message, you must verify that the decoded JWSPayload is bit-for-bit identical to the JSON payload passed in the webhook request. You can use a fast comparison operation, such as a byte (or ordinal string) comparison.

OAuth 1.0
If OAuth 1.0 authentication is configured, the webhook request contains OAuth 1.0 authentication data. The receiving system must validate the request using the shared OAuth configuration agreed during integration setup.

OAuth 2.0
If OAuth 2.0 authentication is configured, the webhook request contains OAuth 2.0 authentication information. The receiving system must validate the bearer token included in the Authorization header using its OAuth 2.0 validation mechanism.

Static Header Validation
ORTEC FS sends the configured headers unchanged with each webhook request. Their interpretation and use are defined by the receiving system.

4.3 Process event

The payload of the webhook request contains a Hypertext Application Language (HAL) compliant reference (link) to the entity representing the object of the event. You can use this link to call the ORTEC FS API to retrieve the latest representation/state of the entity.

5 Retry policy

When making POST calls to your webhook endpoint, ORTEC FS implements a retry mechanism to ensure that the notification reaches your application despite any transient communication issues. This chapter describes some key aspects related to the retry mechanism:

- If a fast response from the webhook endpoint can't be delivered, it's recommended to store the event and perform the processing asynchronously.
- A retry is attempted for any failed request. A request is considered failed in case of a time-out (no response) or if the response status code is different from 200, 201, or 202.
- For each failed request, ORTEC FS retries the call:
  - In the first 30 minutes, the failed request is retried with increasing delays between attempts (exponential backoff strategy).
  - After the first 30 minutes, the failed request is retried every 30 minutes for a maximum of 24 hours.

Contact information

For further information contact ORTEC, either through your existing ORTEC representative or by using the appropriate contact details listed on www.ortec.com

Our website offers case studies, white papers, brochures, demos and much more.