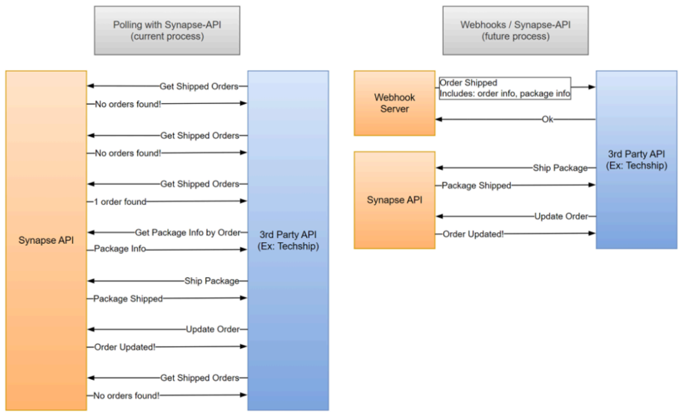
Synapse Webhooks — Usage Guide

Overview

Webhooks allow external systems to receive real-time notifications when key events occur within the Synapse WMS platform (e.g., order creation, load closure, shipment events). A webhook subscription lets a client register an endpoint (URL) to receive these notifications via HTTPS `POST` requests containing structured JSON payloads.

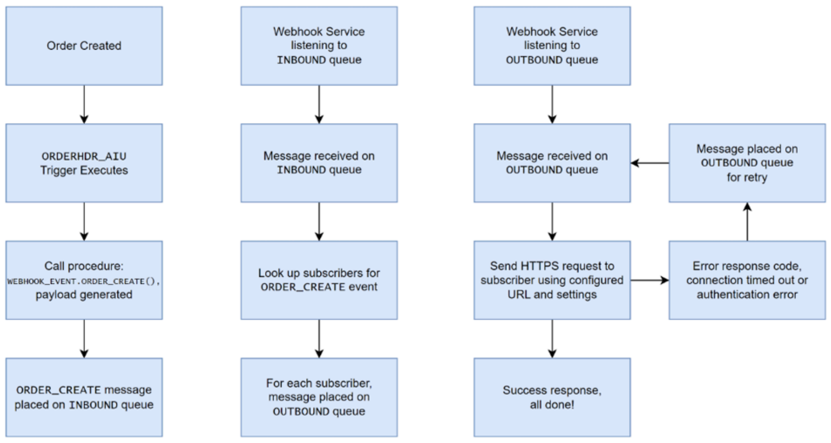
Webhooks are delivered **at least once**, and endpoints must be idempotent to handle possible retries or duplicates.



Example for a TechShip package shipping workflow.

How It Works

1. **Event Triggered:**
   A webhook event (e.g., `ORDER_CREATE`, `LOAD_CLOSE`) is generated by Synapse through database triggers or application hooks.
2. **Inbound Queue:**
   The event payload is placed on the `AQ_WEBHOOK_INBOUND` Oracle queue.
3. **Processing:**
   The Webhook Service (Node.js microservice) listens to inbound events, matches them against active subscriptions, and places messages for each subscriber onto the `AQ_WEBHOOK_OUTBOUND` queue.
4. **Delivery:**
   The Webhook Service then sends HTTPS `POST` requests to subscriber endpoints with event payloads.
5. **Retries & Failures:**
   If delivery fails, the system retries up to 10 times over 24 hours following an exponential backoff schedule.
6. **History:**
   All inbound and outbound transactions are logged and visible for 30 days.


Order Created
ORDERHDR_AIU Trigger Executes
Call procedure: WEBHOOK_EVENT.ORDER_CREATE(), payload generated
ORDER_CREATE message placed on INBOUND queue
Webhook Service listening to INBOUND queue
Message received on INBOUND queue
Look up subscribers for ORDER_CREATE event
For each subscriber, message placed on OUTBOUND queue
Webhook Service listening to OUTBOUND queue
Message received on OUTBOUND queue
Send HTTPS request to subscriber using configured URL and settings
Success response, all done!
Message placed on OUTBOUND queue for retry
Error response code, connection timed out or authentication error

Example ORDER_CREATE webhook workflow from event creation to inbound / outbound processing

---

Subscription Lifecycle

Each subscription is valid for **30 days** and must be renewed periodically.

Subscriptions may have one of the following states:

- **ACTIVE** – Receiving events normally.
- **PENDING** – Awaiting successful PING verification.
- **EXPIRED** – Expired; must be renewed to reactivate.
- **BROKEN** – Marked due to repeated delivery failures.
- **DISABLED** – Manually or automatically disabled.

At creation or update, a **PING** event is sent to verify connectivity before activation.

Authentication & Security

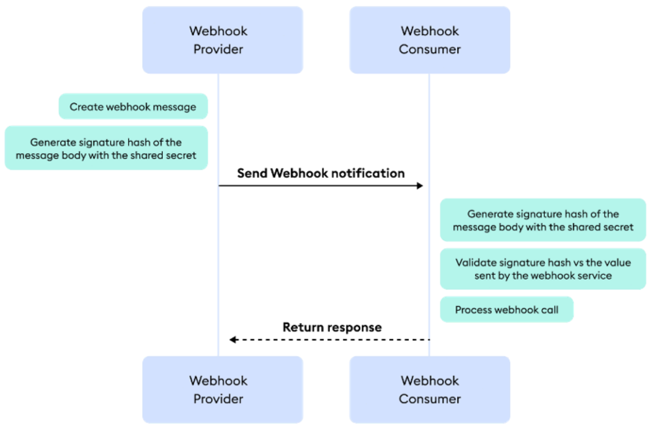
Synapse supports multiple authentication methods when sending requests to subscriber endpoints:

Type	Description
`<null>`	No authentication will be used. This can be useful if an API token or key is passed as a URL parameter.
`BASIC`	Sends HTTP Basic Auth header using client-provided username/password.
`BEARER_TOKEN`	Adds an `Authorization` header with a bearer token (default prefix "Bearer").
`API_KEY`	Includes an API key either in header, query parameter, or cookie.

Optionally, clients can configure a **secret** to cryptographically sign payloads.

Each request includes an `X-SYNAPSE-SIGNATURE` header containing an HMAC-SHA256 signature of the payload.

Clients should verify the signature using their stored secret to confirm authenticity.


Webhook Provider
Webhook Consumer
Create webhook message
Generate signature hash of the message body with the shared secret
Send Webhook notification
Generate signature hash of the message body with the shared secret
Validate signature hash vs the value sent by the webhook service
Process webhook call
Return response
Webhook Provider
Webhook Consumer

Example SHA-256 HMAC validation workflow

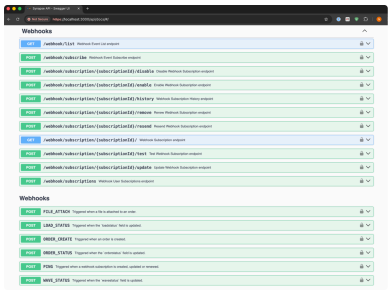
Outbound Request Headers

Each outbound request includes the following headers:

Header	Description
`X-SYNAPSE-REQUEST-ID`	Unique ID for this specific HTTP request attempt.
`X-SYNAPSE-EVENT-ID`	Unique ID representing the event instance.
`X-SYNAPSE-EVENT-TYPE`	The event type (e.g., `ORDER_CREATE`).
`X-SYNAPSE-SUBSCRIPTION-ID`	The ID of the subscription receiving the event.
`X-SYNAPSE-SIGNATURE`	HMAC-SHA256 signature of the payload (if secret provided).

Management API Endpoints

All endpoints exist under `/webhook/` in Synapse API. Please review the Synapse API usage or reference guide for additional details on these endpoints.



Webhooks

GET /webhook/list Webhook Event List endpoint

POST /webhook/subscribe Webhook Event Subscribe endpoint

POST /webhook/subscription/{subscriptionId}/disable Disable Webhook Subscription endpoint

POST /webhook/subscription/{subscriptionId}/enable Enable Webhook Subscription endpoint

POST /webhook/subscription/{subscriptionId}/history Webhook Subscription History endpoint

POST /webhook/subscription/{subscriptionId}/remove Renew Webhook Subscription endpoint

POST /webhook/subscription/{subscriptionId}/resend Resend Webhook Subscription endpoint

GET /webhook/subscription/{subscriptionId}/ Webhook Subscription endpoint

POST /webhook/subscription/{subscriptionId}/test Test Webhook Subscription endpoint

POST /webhook/subscription/{subscriptionId}/update Update Webhook Subscription endpoint

POST /webhook/subscriptions Webhook User Subscriptions endpoint

Webhooks

POST FILE_ATTACH Triggered when a file is attached to an order.

POST LOAD_STATUS Triggered when the `loadstatus` field is updated.

POST ORDER_CREATE Triggered when an order is created.

POST ORDER_STATUS Triggered when the `orderstatus` field is updated.

POST PING Triggered when a webhook subscription is created, updated or renewed.

POST WAVE_STATUS Triggered when the `wavestatus` field is updated.

Synapse API Swagger UI showing Webhooks endpoints and payload schemas

---

List Available Events

`GET /webhook/list`

Returns all supported webhook event types.

List Subscriptions

`POST /webhook/subscriptions`

Lists all subscriptions belonging to the current API user.

Create Subscription

`POST /webhook/subscribe`

Request Body:

```json
{
  "eventType": "ORDER_CREATE",
  "url": "https://example.com/webhooks",
  "headers": { "X-Custom-Header": "ABC123" },
  "filter_custid": "CUST01",
  "filter_facility": "ABC",
  "filter_status": "6",
  "authType": "BEARER_TOKEN",
  "authOptions": { "token": "example_token" },
  "secret":
"0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef"
}
```

Response:

```json
{
  "status": "OKAY",
  "message": null,
  "subscriptionId": "019a2679-10c3-71ef-b36f-d25e78e0d9f8"
}
```

A `PING` event is automatically dispatched to validate the URL. If it fails, the subscription will be marked as **BROKEN**.

If this occurs, after fixing the issue you can call the `update` endpoint which will retry the `PING` event.

Get Subscription Details

`GET /webhook/subscription/:subscriptionId/`

Sensitive fields such as secrets or passwords are masked (returned as `<hidden>`).

Update Subscription

`POST /webhook/subscription/:subscriptionId/update`

Updates a subscription and resets its expiration date 30 days into the future. Triggers a new `PING` event to verify connection.

Enable Subscription

`POST /webhook/subscription/:subscriptionId/enable`

Enables a disabled subscription.

Disable Subscription

`POST /webhook/subscription/:subscriptionId/disable`

Disables a webhook subscription.

Renew Subscription

`POST /webhook/subscription/:subscriptionId/renew`

Extends expiration without modifying configuration.

Test Subscription

`POST /webhook/subscription/:subscriptionId/test`

Sends a simulated `PING` event to the configured endpoint. Useful for connectivity validation.

View Subscription History

`POST /webhook/subscription/:subscriptionId/history`

Retrieves all past delivery attempts for that subscription.

Resend a Request

`POST /webhook/subscription/:subscriptionId/resend`

Allows resending a specific historical event (within 30 days).

```
{
  "requestId": "019a2679-10c3-71ef-b36f-d25e78e0d9f8"
}

```

Expiration & Renewal

- Subscriptions expire **30 days** after creation or last renewal.
- A renewal resets the timer.

- Seven days before expiration, a warning message is logged to the Synapse application messages log (`appmsgs`).
- Expired subscriptions automatically transition to **DISABLED**. A message will be logged to the Synapse application messages log (`appmsgs`).

Retry Policy

If a subscriber's endpoint fails to respond successfully:

- The request is retried up to **10 times over 24 hours**.
- The retry schedule is progressive (5, 10, 15, 30, 60, 120, 240, 480, 480 minutes).
- After 10 failures, the subscription is marked **BROKEN**.
- Successful delivery resets the failure counter to 0.

Event Delivery Model

- **Protocol:** `HTTPS`
- **Method:** `POST`
- **Content-Type:** `application/json`
- **Timeout:** 60 seconds
- **Redirects:** Limited to prevent loops
- **User-Agent:** `Synapse/Webhook-Service`

Payload Format

Payloads are in `JSON` format. Each event type has its own predefined JSON schema, accessible via Synapse API. The schema defines the payload's structure, fields, and data types.

Example `ORDER_CREATE` payload:

```
[
  {
    "synapse_environment": "TEST",
    "orderid": 123456,
    "shipid": 1,
    "customer": "TESTCUST",
    "order_type": "O",
    "order_type_desc": "Outbound",
    "po": "TEST PO",
    "reference": "TEST REF",
    "from_facility": "ABC",
    "loadno": 654321,
    "load_stopno": 1,
    "load_stop_shipno": 1,
    "order_detail": [
      {
        "orderid": 123456,
        "shipid": 1,
        "item": "TESTITEM",
        "customer": "TESTCUST",
        "unit_of_measure": "EA",
        "quantity": 5
      }
    ]
  }
]
```

(this is a subset of the actual payload, please review the Synapse API documentation for the full schema)

Duplicate Delivery Handling

Because delivery is **at least once**, your endpoint should:

1. Log received `eventId` s.
2. Skip processing for any previously seen event ID.

This ensures idempotency and prevents double-processing.

Example HMAC Verification (JavaScript)

```javascript
import crypto from 'crypto';

function verifySignature(secret, headers, body) {
  const { 'x-synapse-signature': signature } = headers;
  const concat = [
    headers['x-synapse-subscription-id'],
    headers['x-synapse-event-id'],
    headers['x-synapse-event-type'],
    headers['x-synapse-request-id'],
    body
  ].join('');

  const expected = crypto.createHmac('sha256', secret)
    .update(concat)
    .digest('hex');

  return crypto.timingSafeEqual(Buffer.from(signature),
Buffer.from(expected));
}

```

Best Practices

- Always use **HTTPS** with a valid certificate.
- Verify **HMAC signatures** for authenticity.
- Use **unique event IDs** to ensure idempotent handling.
- Log both **successes and failures** for observability.
- Keep your endpoint's response times below **60 seconds**.
- Handle **duplicate** deliveries gracefully.

Logging & History Retention

- Both inbound and outbound webhook activity is logged for **30 days**.
- Logs include event IDs, request/response metadata, and payloads.
- The `WEBHOOK_PURGE_HISTORY` job removes records older than 30 days automatically.