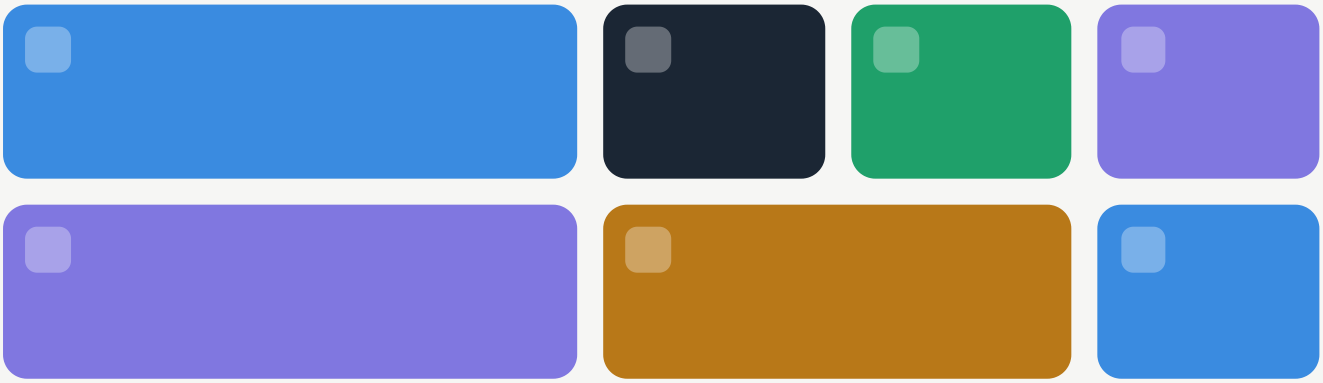
BPMSquare

CUSTOMER EDITION

Change feed and webhooks

How to connect other systems to BPMSquare: the REST API, the change feed and webhooks, connectors, the integration middleware and Data Workbench.

New way of customer relationship



Change feed and webhooks

Provided by **Snasquare**

Powered by **BPMSquare** · Snapshot 2026-10-29

CHAPTER 3

Change feed and webhooks

> Screen names and button words may be refined as the product improves. If a name here is slightly different on your screen, look for the same idea nearby.

About this chapter

This chapter is for developers who need to keep another system in step with BPMSquare. There are two ways to learn that something changed. You can ask: that is the **change feed**. Or you can be told: that is a **webhook**. BPMSquare can also send one record to another system on demand, and an ERP can tell BPMSquare that an order changed.

Read Chapter 1 for keys and limits. The examples use made-up data: Example Trading LLC, a quotation `QT-2026-0042`, and a receiver at `hooks.example.com`.

Contents

1. Ask or be told
2. The change feed
3. Event webhooks
4. Verify the signature
5. Delivery, retries and idempotency
6. Call webhooks: one record when something happens
7. Push to ERP (the older way)
8. Telling BPMSquare that an ERP order changed
9. Check that it works

1. Ask or be told

	Change feed	Event webhook
How it works	You call `GET /changes` with a cursor.	BPMSquare calls your address with a signed batch.
Who sets the pace	You.	BPMSquare's delivery schedule.
Needs a public address	No.	Yes, an `https` address.

	Change feed	Event webhook
Access control	The key's scopes decide what you see.	An administrator chooses the objects. No key is involved.
Good for	A job that runs every few minutes, a first load, catching up after downtime.	Letting your system react without polling.
Missed changes	None. Keep your cursor and ask again.	None. A failed batch is sent again.

Both read the same record: the workspace's change history. An event in a webhook has the same shape as an item in the change feed.

Webhooks are delivered in batches on a schedule, not the moment a record changes. If you need to know within seconds, poll the change feed. Ask your BPMSquare contact how often deliveries run for your workspace.

A common pattern is to use both: a webhook as the prompt that "something changed", and the change feed as the source of truth you read after it.

---

2. The change feed

`GET /api/v1/changes` is an ordered log of every create, update and delete in your workspace. It is built for synchronisation: you keep a cursor, and each call returns what happened after it.

The cursor is based on the time and id of the last item. No item is skipped or repeated across pages, even when two changes happen in the same instant.

The key needs read access. The feed shows only the object types that the key could read through the list endpoints. A key limited to `quotations` sees quotation changes only.

Parameters

Parameter	Notes
`since`	The `cursor` from a previous answer. Leave it out to start at the beginning of the log. A cursor that was not returned by this endpoint is a `422`.
`object_type`	Only one object type, for example `quotes`. `403` if the key cannot read that type.
`limit`	Defaults to 100. Maximum 500.

```
curl -s -H "Authorization: Bearer $KEY" "$BASE/changes?object_type=quotes&limit=100"
```

```
{
  "data": [
    {
      "id": "7a1b2c3d-4e5f-4a6b-8c7d-9e0f1a2b3c4d",
      "object_type": "quotes",
      "object_id": "b7d1e0c2-5a3f-4e18-8c92-6f4a1d9e2b03",
      "object_label": "QT-2026-0042",
      "action": "create",
      "changes": [
        { "field": "account_id", "from": null, "to": "3f2a9c1e-7b44-4d0a-9a51-0c6d2e8b1a01" },
        { "field": "total", "from": null, "to": 41050 }
      ],
      "actor": "api:v1",
      "at": "2026-10-26T06:12:31.502114+00:00"
    },
    {
      "id": "8b2c3d4e-5f60-4b7c-9d8e-0f1a2b3c4d5e",
      "object_type": "quotes",
      "object_id": "b7d1e0c2-5a3f-4e18-8c92-6f4a1d9e2b03",
      "object_label": "QT-2026-0042",
      "action": "update",
      "changes": [ { "field": "status", "from": "draft", "to": "sent" } ],
      "actor": "sara.admin@example.com",
      "at": "2026-10-27T09:41:12.318204+00:00"
    }
  ],
  "meta": { "count": 2, "has_more": false, "object_type": "quotes" },
  "cursor": "MjAyNi0xMC0yN1QwOTo0MToxMi4zMTgyMDQrMDA6MDB8OGIyYzNkNGUtNWY2MC00YjdjLTlkOGUtMGYxYTJiM2M0ZDVl",
  "_links": {
    "self": "/api/v1/changes",
    "next": "/api/v1/changes?
since=MjAyNi0xMC0yN1QwOTo0MToxMi4zMTgyMDQrMDA6MDB8OGIyYzNkNGUtNWY2MC00YjdjLTlkOGUtMGYxYTJiM2M0ZDVl&object_ty
pe=quotes&limit=100"
  }
}
```


What an item holds

Field	Meaning
`id`	The id of this change. Unique. Use it to remove duplicates.
`object_type`	The kind of record, such as `quotes` or `accounts`.
`object_id`	The record's id. Match on this, not on `object_label`.
`object_label`	The record's reference or name at the time.
`action`	`create`, `update`, `delete` or `reopen` (a quotation moved back into the pipeline).
`changes`	A list of `{ field, from, to }`.
`actor`	The person's email, or `api:v1` for a change made through the API.

Field	Meaning
`at`	When it happened.

Things to know about `changes`:

- Encrypted personal fields (account and contact phone, email, tax number, and staff phone and email) show `"[redacted]"` with `"redacted": true`. The change is visible. The value is not.
- Changes to lines appear as text, such as `Line 1`, `Line 2 added`, `Line 3 removed`. The order of lines is by position.
- The cost price of a product is left out of product changes.
- A `delete` tells you the record is gone. Fetch nothing. Remove it on your side.

Object types

These are the types the application logs: `accounts`, `contacts`, `quotes`, `cases`, `invoices`, `purchase_orders`, `inventory`, `products`, `employees`, `rfqs`, `estimates`, `opportunities`, `wfm_projects`, `standard_quotes`, `sales_orders`, `abm_programs`, `abm_target_accounts`, and more.

A key needs the matching scope to see a type:

Object type	Scope the key needs
`quotes`	`quotations`
`purchase_orders`	`purchase-orders`
`wfm_projects`	`projects`
`abm_target_accounts`	`abm_targets` (named)
`accounts`, `cases`, `invoices`, `inventory`, `products`, `employees` (named), `rfqs` (named), `estimates` (named), `opportunities`, `standard_quotes`, `sales_orders`, `abm_programs`	The scope of the same name

Types with no scope of their own, such as assets, suppliers, work orders and leads, are visible only to a key with **All objects**.

Reading the feed correctly

1. **First load.** Call without `since` and follow `cursor` until `meta.has_more` is false.
2. **Normal run.** Call with the `cursor` you stored last time.
3. **Save the cursor only after you have processed the page.** If your job fails half-way, you read the page again. That is safe if you remove duplicates on `id`.
4. **An empty page.** When nothing is returned, `cursor` repeats your `since` (or is `null` if you sent none). Keep it.

5. **A full last page.** `has_more` is true when the page is full (`count` equals `limit`). A final full page can say `true` and the next call can return nothing. That is normal.

Errors: `401`, `403` (an `object_type` outside the key's scope), `422 {"error":"Invalid cursor","message":"` since `must be a cursor returned by a previous /changes response."}`, `500`.

The change feed is a log, not a copy of the data. To get the current record, read it with its list or detail endpoint using `object_id`. Chapter 7 shows a full loop.

---

3. Event webhooks

An Event webhook sends the same items as the change feed, in signed batches, to an address you give.

Register one

You must be a workspace administrator.

1. Go to **Settings > Integrations > Webhooks**. If your workspace does not show that entry, go to **Settings > Integrations > Webhooks & push** instead. Both do the same job for events. The first is a newer, single list. Ask your BPMSquare contact which one your workspace has.
2. Press **New webhook**. Type a **Name**. Choose **Event** as the **Kind**.
3. Under **Changes sent**, tick **All objects**, or tick the objects you want: Quotes, Accounts, Contacts, Cases, Work orders, Invoices, Purchase orders, Inventory, Assets, Suppliers.
4. On the **Request** tab, type the **Address**. It must start with `https`. An address on a private or internal network is refused, both when you save it and each time BPMSquare calls it.
5. Press **Create**. Open the **Security** tab and copy the **Signing secret**. It looks like `whsec_` followed by 48 hexadecimal characters. It is shown once. Store it in your own secret store.

A new webhook sends only changes made after it was created. It does not replay history. To bring in history, read the change feed once.

Choosing **All objects** sends every change type recorded in your workspace's change history, including staff records. A webhook is not limited by a key's scopes. Tick only the objects the receiver should see.

In the older **Webhooks & push** page the steps are shorter: **+ New webhook**, a **Name**, an **Endpoint URL**, **All objects** or a list, then **Create webhook**. **Pause**, **Resume** and **Delete** are on each row.

What is sent

BPMSquare sends a `POST` with a JSON body. The call times out after 10 seconds.

```
Content-Type: application/json
X-BPMSquare-Signature: <hex HMAC-SHA256 of the raw body, key = your whsec_ secret>
X-BPMSquare-Delivery: <delivery_id>
```

```
{
  "delivery_id": "0f1e2d3c-4b5a-4968-8777-665544332211",
  "sent_at": "2026-10-28T20:30:04.118Z",
  "events": [
    {
      "id": "8b2c3d4e-5f60-4b7c-9d8e-0f1a2b3c4d5e",
      "object_type": "quotes",
      "object_id": "b7d1e0c2-5a3f-4e18-8c92-6f4a1d9e2b03",
      "object_label": "QT-2026-0042",
      "action": "update",
      "changes": [ { "field": "status", "from": "draft", "to": "sent" } ],
      "actor": "sara.admin@example.com",
      "at": "2026-10-27T09:41:12.318204+00:00"
    }
  ]
}
```

Each item in `events` has the same shape as a change-feed item. One batch holds up to 100 events, oldest first.

The test message

Test (on the webhook's Test tab, or on the older page) sends a signed message marked as a test. The older page sends the normal batch shape with one event that has `object_type: "ping"` and `action: "ping"`. The newer page sends a flat body: `{ "event": "webhook.test", "delivery_id": …, "sent_at": …, "workspace": …, "ping": true }`. Accept it, check the signature, and do not treat it as a data change. A test on the newer page never changes the webhook's failure count.

---

4. Verify the signature

Every webhook call is signed. Check the signature before you trust the body.

- The signature is the hexadecimal HMAC-SHA256 of the **raw request body**.
- The key is the webhook's signing secret, exactly as shown, including the `whsec_` prefix.
- It is sent in the `X-BPMSquare-Signature` header.
- Compare in constant time.
- Compute it over the raw bytes. If you parse the JSON first and serialise it again, the bytes can differ and the check fails.

Node (Express)

```
import crypto from "node:crypto";
import express from "express";

const app = express();
const SECRET = readSecret("bpmsquare_webhook_secret"); // your own secret store; the whsec_… value

app.post("/hooks/bpmsquare", express.raw({ type: "application/json" }), (req, res) => {
  const given = req.get("X-BPMSquare-Signature") || "";
  const expected = crypto.createHmac("sha256", SECRET).update(req.body).digest("hex");
  const ok =
    given.length === expected.length &&
    crypto.timingSafeEqual(Buffer.from(given), Buffer.from(expected));
  if (!ok) return res.status(401).send("bad signature");

  const { delivery_id, events } = JSON.parse(req.body.toString("utf8"));
  for (const e of events) {
    // Remove duplicates on e.id, then handle e.object_type, e.action and e.changes.
  }
  res.sendStatus(200); // any 2xx acknowledges the batch
});

app.listen(3000);
```

Python (Flask)

```
import hashlib
import hmac
import json

from flask import Flask, abort, request

app = Flask(__name__)
SECRET = read_secret("bpmsquare_webhook_secret").encode("utf-8")  # your own secret store; the whsec_… value


@app.post("/hooks/bpmsquare")
def bpmsquare_hook():
    raw = request.get_data()  # the raw bytes, before any parsing
    given = request.headers.get("X-BPMSquare-Signature", "")
    expected = hmac.new(SECRET, raw, hashlib.sha256).hexdigest()
    if not hmac.compare_digest(given, expected):
        abort(401)

    batch = json.loads(raw)
    for event in batch["events"]:
        pass  # Remove duplicates on event["id"], then handle the change.
    return "", 200  # any 2xx acknowledges the batch
```

`read_secret` stands for however your company keeps secrets. Do not put the secret in source code.

If the signature is wrong, answer `401` and do nothing with the body. Do not tell the sender why.

---

5. Delivery, retries and idempotency

How an Event webhook is delivered:

- **Batches.** Up to 100 events per delivery, oldest first. A larger backlog is delivered over several deliveries.
- **On a schedule.** Deliveries are not instant. Ask your BPMSquare contact how often they run for your workspace.
- **Success.** Any `2xx` answer is a success. BPMSquare then moves its position forward.
- **Failure.** Any other status, a timeout (10 seconds) or a connection error is a failure. BPMSquare does not move forward. The same events are sent again at a later delivery, with a **new** `delivery_id`.
- **Pausing.** After 15 failures in a row the webhook pauses itself. The list shows it as paused. An administrator presses **Resume** (or switches it on) to start it again. It carries on from where it stopped, so nothing is lost.
- **Order.** Events within a batch are in time order. A retry sends the oldest undelivered events first.

What this means for your receiver:

1. **Answer fast.** Acknowledge within 10 seconds. Put the work on a queue and answer `200` straight away.
2. **Expect repeats.** A batch can arrive again, for example when your answer was lost. Make the handler idempotent on `events[].id`. Keep the ids you have processed, and skip any you have seen.
3. **Do not rely on `delivery_id`.** It changes on every attempt. Use the event `id`.
4. **Do not rely on one event per call.** Always loop over `events`.
5. **Be careful with order across batches.** Apply an event only if its `at` is later than the last one you applied for that `object_id`.
6. **Treat the event as a prompt.** For the current state of a record, read it with its endpoint.
7. **Answer `401` only for a bad signature.** Answer `5xx` when you are down, so the batch is sent again. Answer `2xx` for anything you accept, even if you choose to ignore it.

---

6. Call webhooks: one record when something happens

A **Call** webhook sends a single record when something fires it. It is available when **Settings > Integrations > Webhooks** is present in your workspace. A Call webhook does not read the change feed. It has no schedule.

What can fire it:

- A **Standard Quote** is awarded (the Integration middleware, Chapter 5, can choose a webhook as the place an awarded quote goes).
- A **button on Accounts**: tick **Button on Accounts** and every account page gets a **Send to this webhook** entry.
- A **Coverage** route that names the webhook for a group of accounts.
- The **Test** tab of the webhook, with a real record.

Set up

1. **Settings > Integrations > Webhooks > New webhook**.

2. On **Basics**, type a **Name** and choose **Call** as the **Kind**.
3. Choose what it **Sends**: **Accounts** or **Standard Quotes**. The kind and the object cannot be changed later.
4. On **Request**, type the **Address**. The address may hold tokens in the path or query, such as `{{account.ref}}`, but never personal data. Choose the **Method**: `GET` (no body), `POST`, `PUT` or `PATCH`.
5. Choose the **Payload**:
   - **Standard**: a signed record in a fixed shape (below).
   - **Template**: your own JSON, with tokens picked from a list.
   - **Body from mapping**: the body built on the Integration middleware Mapping tab.
6. Add the **Headers** your receiver expects, such as an API key. There can be at most 10. After you save, a header value is hidden and can only be replaced.
7. Leave **Send personal data** off unless the receiver needs email, phone or tax numbers.
8. Set **Wait for an answer (seconds)**. The default is 15.
9. Optionally set **Read the id from the answer**, such as `$.data.number`. BPMSquare keeps that number against the record.
10. Press **Create**. On **Security**, copy the **Signing secret** (shown once). On **Test**, choose a real record and press **Preview** to see the exact request without sending, or press the send button.

What is sent

The Standard payload for an account:

```
{
  "event": "account.pushed",
  "delivery_id": "6c1d9e0a-3b52-4f7e-a8d1-2e4b6c8d0f13",
  "sent_at": "2026-10-28T09:00:00.000Z",
  "workspace": "Example Trading LLC",
  "record": {
    "id": "3f2a9c1e-7b44-4d0a-9a51-0c6d2e8b1a01",
    "url": "https://bpm.example.com/accounts/3f2a9c1e-7b44-4d0a-9a51-0c6d2e8b1a01",
    "account": { "ref": "ACC-0007", "name": "Example Trading LLC", "type": "end_customer", "city": "Dubai",
"country": "AE" }
  }
}
```

The `record` carries the object's allowed fields. Email, phone and tax numbers are left out unless **Send personal data** is on. The event name is `account.pushed` for an account, `quote.awarded` for an awarded Standard Quote, and `record.sent` for anything else. A test uses `webhook.test`. This sample is shortened.

Headers added to every Call webhook. Your own headers cannot replace them:

Header	Value
`X-BPMSquare-Signature`	Hex HMAC-SHA256 of the raw body (an empty body for `GET`). Key: the signing secret.
`X-BPMSquare-Delivery`	A new id for each attempt.
`X-BPMSquare-Event`	The event name.

Header	Value
`X-BPMSquare-Timestamp`	Seconds since 1 January 1970, when the call was made. The signature does not cover it.
`X-BPMSquare-Idempotency-Key`	A key that stays the same for every attempt at the same firing, including a restart from the Monitor. Use it to ignore repeats.

Retries

If the receiver answers `429` or `5xx`, or the connection fails, BPMSquare waits 2 seconds and tries once more. Any other `4xx` is not repeated. If it still fails, the call is shown as **Failed** in the Integration middleware Monitor with a plain-language reason. Press **Restart** there to send it again with the same idempotency key. A success settles the earlier failed lines for that record.

A Call webhook that fails 15 times in a row pauses itself. **Resume** starts it again. A test never changes the failure count.

A webhook that a link, a button or a Coverage route uses cannot be deleted. The panel says where it is used.

The **Log** tab lists the last 20 calls. The Monitor keeps their details for two days.

---

7. Push to ERP (the older way)

If your workspace does not have the Call webhooks above, an account page has a **Push to ERP** action. It sends one account to the **Receiving URL** you set under **Settings > Integrations > Webhooks & push**.

1. Under **Push to external systems**, type the **Receiving URL**. It must be `https`.
2. Under **Signing secret**, press **Generate**. Copy the value at once with **Copy**. It is shown only then. **Regenerate** replaces it and the old one stops at once.
3. Press **Save**.

The body is:

```
{
  "event": "account.pushed",
  "data": { "id": "3f2a9c1e-7b44-4d0a-9a51-0c6d2e8b1a01", "name": "Example Trading LLC", "phone": "+971 4
555 0147", "email": "procurement@example.com" },
  "sent_at": "2026-10-28T09:00:00.000Z"
}
```

`data` is the account record, including its phone, email and tax number. It is shortened here. The call has a 15-second timeout. If a signing secret is set, the call carries `X-BPMSquare-Signature`, signed the same way as above. Without a secret the push is not signed, so always set one.

Verify it exactly as in section 4. Only the shape of the body is different: there is no `events` list. There is no automatic retry. A failure is reported on the screen, and the person presses **Push to ERP** again.

---

8. Telling BPMSquare that an ERP order changed

When your workspace reads sales orders from Dynamics 365 or SAP S/4HANA, BPMSquare refreshes them on a schedule. If you want an order refreshed now, your ERP can tell BPMSquare. This needs the **Sales Orders** and **Integration middleware** modules, and an ERP connection (Chapter 5).

```
POST https://<your-workspace-domain>/api/middleware/webhook/sales-order
Content-Type: application/json
X-BPMSquare-Signature: <hex HMAC-SHA256 of the raw body>
```

```
{ "order_ref": "SO-0004", "erp_id": "0000451122", "sent_at": "2026-10-28T09:00:00.000Z" }
```

Field	Meaning
`order_ref`	The ERP's order number, as shown on the order's ERP line.
`erp_id`	The ERP's own record id. A GUID for Dynamics 365, the sales order number (digits) for SAP.
`sent_at`	Optional. A time more than 10 minutes away from now is refused.

Send `order_ref`, `erp_id`, or both. Keep the body under 8 KB.

The signing key is the **Signing secret** under **Settings > Integrations > Webhooks & push**, the same secret that signs pushes to your ERP. The call is not signed with an API key. If your workspace shows only the newer Webhooks list and you do not see that page, ask your BPMSquare contact to help you set up the secret.

The call goes to your workspace's own address. The body never names a workspace.

Answers:

Answer	Meaning
`200 {"ok":true,"throttled":false,"invoices":1}`	The order and its documents were refreshed. `throttled: true` means it was refreshed in the last 10 seconds and nothing was done.
`400`	The body has neither `order_ref` nor `erp_id`, or is not JSON.
`401`	A bare refusal. The signature, the address or the modules are wrong. The answer does not say which.
`404 {"ok":false,"error":"That order is not known to this workspace."}`	No order matches.
`502`	The ERP could not be read. Try again later.
`503`	A database update is waiting in your workspace. Ask your BPMSquare contact.

An order that the ERP names but BPMSquare does not have yet is read in, if the **Sales orders** link is switched on (Chapter 5).

A Python call:

```
import hashlib
import hmac
import json
import requests

BASE = "https://bpm.example.com"
secret = read_secret("bpmsquare_erp_signing_secret").encode("utf-8")  # your own secret store

body = json.dumps({"order_ref": "SO-0004", "erp_id": "0000451122"}).encode("utf-8")
signature = hmac.new(secret, body, hashlib.sha256).hexdigest()
resp = requests.post(
    f"{BASE}/api/middleware/webhook/sales-order",
    data=body,
    headers={"Content-Type": "application/json", "X-BPMSquare-Signature": signature},
    timeout=30,
)
print(resp.status_code, resp.json())
```

Sign the exact bytes you send.

---

9. Check that it works

- **Event webhook.** Use **Test**. Your receiver must verify the signature and answer `2xx`. The webhook's row shows its last call, its state and its last error. Then change a record and wait for the next delivery.
- **Call webhook.** Use **Preview** to see the request, then send a real record from the **Test** tab. The **Log** tab shows the answer. The Integration middleware **Monitor** (Chapter 5) shows each call as one line, with **View request** and **View response**.
- **Change feed.** Make a change in the app, then call `GET /changes` with your stored cursor. The change appears with your name as `actor`.
- **Signature.** Send a body with one wrong byte. Your receiver must answer `401`.
- **Repeats.** Send the same batch twice. Your receiver must do the work once.