ABLE CARE

Able Platform: Webhook integration

29 September, 2025

1 Overview

Webhooks are the primary way to integrate partner systems with Able Platform. Whenever new measurements are recorded, the platform pushes these events to a configured endpoint. This avoids the need for polling APIs, reduces integration complexity, and ensures partner systems always have the latest data available.

2 Webhook Registration

Our platform emits two types of events for partner systems:

- `measurement.recorded` : emitted when an individual completes a measurement (e.g. grip strength, gait speed, etc.)
- `screening.recorded` : emitted when a screening session is completed, consisting of a set of measurements.

Every organization can register webhooks, by adding/configuring a new webhook endpoint in our system. An optional API Key for the receiving webhook endpoint can be configured in a separate field. If an API Key is configured for the endpoint, it will be used as the value for the `X-API-Key` HTTP header. Our system can optionally include the API Key in an `Authorization` header, for this you need to include the `auth-scheme` (i.e. `Bearer` , `Token` , etc.) in the relevant field.

Currently registration will apply to all webhook events. In the future, as we add more events, we will enable configuring the webhooks to only a subset of events.

The registration process is simple and comprises:

- Logging into the Admin Dashboard for you org.
- Updating the "Webhook" object in the sidebar by:
  - Inserting the receiving webhook URL
  - Amending other optional fields (e.g. access key, custom header, etc.)

Access to the Admin dashboard (URL and credentials) will be sent to you via an automated onboarding email.

Authentication is at the discretion of the partner. It is possible to configure HTTP receiving endpoints for development purposes, and authentication headers *will be skipped* if no secret value is provided.

3 Webhooks

3.1 Webhook Event Payload

```
{
  "id": "evt_123abc",
  "type": "measurement.recorded | screening.recorded",
  "created": "2025-09-09T10:45:30Z",
  "version": "2025-09-24",
  "organization": "org_123abc",
  "data": "<Measurement> | <Screening>"
}
```

3.1.1 Fields:

- `id` : Unique identifier for the webhook event (used for deduplication).
- `type` : The type of event; currently either `measurement.recorded` or `screening.recorded` .
- `created` : ISO 8601 timestamp of when the event was created on the Able Platform.
- `version` : Version of the webhook payload schema to support backward compatibility. The version is based on Calver.
- `organization` : Identifier of organization the data belongs to.
- `data` : The payload with either a Measurement or a Screening object depending on the event type.

3.2 Screening

```
{
  "id": "scrn_123abc",
  "type": "falls",
  "created": "2025-09-09T10:45:30Z",
  "at_risk": "true|false|null",
  "measurements": ["<Measurement>, <Measurement>..."]
}
```

3.2.1 Fields:

- `id` : The unique identifier of the screening session.

- `type` : The screening protocol name. Currently, only “falls”.
- `created` : When the screening session was completed (ISO 8601, UTC).
- `at_risk` : Overall risk outcome for the screening; `true` , `false` , or `null` if not determined.
- `measurements` : A list of Measurements that were taken as part of this specific screening.

3.3 Measurement

```
{
  "id": "meas_123abc",
  "type": "smgt" | "tug" | "4gs" | "cst",
  "result": "12.34",
  "unit": "m | m/s | reps | kg",
  "at_risk": "true|false|null",
  "values": ["<Value>, <Value>..."] | null,
}
```

3.3.1 Fields:

- `id` : The unique identifier of the measurement.
- `type` : The measurement code indicating the test performed (e.g., smgt, tug, 4gs, cst).
- `result` : The primary outcome value for the measurement as recorded by the platform.
- `unit` : The unit corresponding to result (e.g., m, m/s, reps, kg).
- `at_risk` : Risk classification derived from this measurement; `true` , `false` , or `null` if not assessed.
- `values` : List of Value objects (e.g., per-hand or per-attempt details) that contributed to the result, or `null` if not applicable.

3.4 Value

```
{
  "hand": "right" | "left" | null,
  "result": "12.34",
  "unit": "kg",
  "created": "2025-09-09T10:45:30Z"
}
```

3.4.1 Fields:

- `hand` : In case of a Grip Measurement (SMGT), the hand this value was recorded on. Can be `right` , `left` , or `null` if 'not applicable.'
- `result` : The numeric result recorded for this specific value.
- `unit` : The unit corresponding to result (e.g., kg).
- `created` : Timestamp when this value was recorded (ISO 8601, UTC). Optional.

4 Webhook Delivery Requirements

Webhook receivers must meet the following technical requirements to ensure reliable delivery and processing of events.

4.1 Transport

- **Method:** Only `POST` requests are used. Other HTTP methods are not supported.
- **Protocol:** All requests are delivered over **HTTPS (TLS 1.2+)**. Plain HTTP endpoints are not accepted in `production` environments (see note on Webhook Registration).
- **Timeout:** Receivers must return a response **within 30 seconds**. After this, the request is considered failed and will be retried.

4.2 Format

- **Content-Type:** All webhook payloads are delivered as `application/json` in UTF-8 encoding.
- **User-Agent:** Requests include a user agent string of the form `Able-Webhooks/{version}` (e.g. `Able-Webhooks/2025-10-03`). This is the webhook implementation version and is different from the schema version.
- **Schema Version:** The `version` field in the payload specifies the payload schema version.

4.3 Security and Privacy

- **Authentication:**
  - If a secret is configured, requests will include an `X-API-Key` header.
  - Optionally, the `Authorization` header can be used with a scheme such as `Bearer <secret>` or `Token <secret>`.
- **Data privacy**: No values included in the payload contain any personally identifiable information.

4.4 Response Handling

- **Success:** Respond with an HTTP `2xx` status if the payload was received and accepted. Processing may continue asynchronously.
- **Error:** Respond with an HTTP `4xx` or `5xx` status to indicate failure (e.g. `400` - Bad Request, `404` - Not Found, `500 - Internal Server Error` etc.) This will trigger a retry.
- **Error responses**: If possible, return a structured JSON error:

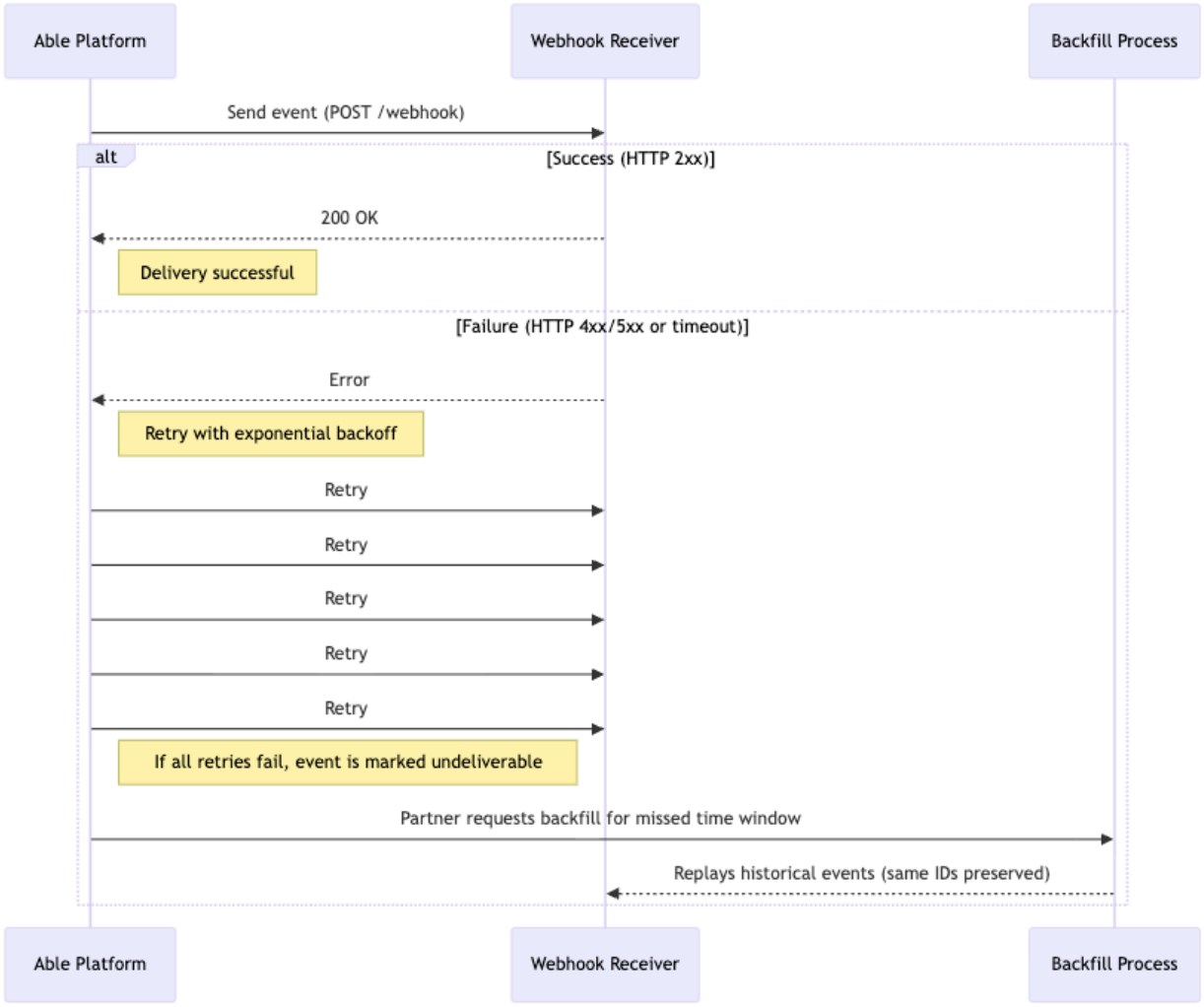
```
{
  "error": {
    "code": "invalid_payload",
    "message": "Description of the error"
  }
}
```

4.5 Idempotence

- Every webhook event includes a unique `id` field.
- The same `id` value always refers to the same payload, and can be used by receivers to safely deduplicate events.

4.6 Delivery & Retry Policy

- **Retries:** Failed requests are retried up to **5 times** within a **24-hour** window.
- **Backoff:** Exponential backoff (e.g. 2s, 4s, 8s, 16s, 32s) with additional temporal jitter applied.
- **Failure Recovery:** After 5 failed attempts, the event is marked as undeliverable. Partners can use the Backfill process to recover missed data.



1 Sequence diagram: Webhook sending, receiving, and backfills

5 Backfill

In some cases, partners may need historical data that predates their webhook subscription (e.g. to backfill an individual's measurements or to synchronize screening records for a given period). To support this, Able Platform will offer a **backfill process**.

Partners can request backfill for a specific organization and a defined time window (e.g. "all events between `2025-08-01` and `2025-09-01`"). Requests are initiated by contacting support@able-care.co.

Backfill can also be used for reconciliation: if your system missed or failed to process webhook deliveries, you can request a replay for the affected time window. Since all events retain their original identifiers, duplicates can be safely ignored during reconciliation.

By default, backfilled events are delivered through the configured webhook endpoint, using the same retry and idempotence policies.

6 Developer Testing

You can use our **staging environment** to validate webhook integrations before going live. Please contact support@able-care.co to set up a staging account and receive onboarding instructions.

In staging you can:

- Register webhook endpoints for your test organization.
- Trigger sample `measurement.recorded` and `screening.recorded` events with synthetic data.
- Validate that your endpoint processes payloads correctly. Payload schemas, headers, retries, and idempotence behavior are identical to production.

Recommendations for testing:

- Use tools like `ngrok` or `localtunnel` to expose a local endpoint for webhook delivery.
- Log all incoming requests to inspect payload structure and headers.
- Verify idempotence by handling duplicate `id` values without side effects.
- Configure and validate authentication headers (e.g., `X-API-Key`) if applicable.
- Simulate failures by returning a non-2xx response to test retries and exponential backoff (2s, 4s, 8s…).
- Request a **backfill replay** to test reconciliation workflows.

All webhook requests in staging can use HTTPS with TLS 1.2+, and no payload contains personally identifiable information.

7 Change Management

We use Calendar-based versioning (e.g. `2025-09-09`). Non breaking changes will be marked as point releases on the version (e.g. `2025-09-09.1`, `2025-09-09.2`, etc.) with the date indicative of when the initial breaking change occurred.

You can expect the following communications in case of changes:

7.1 Non-breaking changes

- **Changes include**:
  - New fields added to existing objects.
  - New event types or possible values added to existing fields.
  - Documentation clarifications or stricter validation rules that remain backward compatible.
- **Notification**: Email notification to registered technical contacts
- **Action needed:** No immediate action.

> Your receiving endpoint should be designed to ignore unknown fields and accept *point* releases. You may optionally update validation logic to take advantage of new fields.

7.2 Breaking changes

- **Changes include**:
  - Removal or renaming of existing fields.
  - Change of data types or semantics of existing fields.
  - Removal of supported event types or modification of delivery guarantees.
- **Notification**: At least **30 days advance notice** via email and release notes.
- **Action needed:**
  - Update your webhook receiver to handle the new schema version.
  - Use the new `version` field in the payload to branch parsing logic if supporting multiple versions in parallel.
  - Complete migration before the deprecation date of the old version.