Stripe Webhooks and the Typed Event Bus

Stripe · https://namazustudios.com/docs/add-ons/commerce/stripe/stripe-webhooks-and-events/

The Stripe Element keeps your platform's view of payments, invoices, and subscriptions in sync with Stripe by receiving webhooks at a single endpoint, verifying their signature, and republishing them as internal Elements events. Every verified webhook is published as a raw event; the most common event types are also published as strongly-typed records so consuming code never has to deserialize a Stripe payload itself.

The Webhook Endpoint

`POST /stripe/webhook` takes the raw request body and a `Stripe-Signature` header, and verifies them with the Stripe SDK's `Webhook.constructEvent` against the configured webhook secret. This endpoint requires no session authentication — Stripe can't supply a session secret — but every request is rejected unless its signature checks out:

- **503** if no webhook secret is configured yet
- **400** if the signature doesn't verify
- **200** `{"received":true}` once the event is published, or `{"received":true,"typed":false}` if the raw event was published but Stripe's payload couldn't be deserialized into a typed record (see below)

On every successfully verified webhook — regardless of type — the endpoint first publishes a `StripeRawEvent` and writes an entry to the event log, *then* attempts to dispatch a typed event. This ordering means a mismatch between this Element's Stripe SDK version and the shape of an incoming event (an `EventDataObjectDeserializationException`) never loses the event — the raw event and audit log entry are already recorded before the typed dispatch is attempted.

Stripe Dashboard Setup

In the Stripe Dashboard, go to **Developers → Webhooks → Add endpoint**.

1. Use the **"Account"** webhook type, not "Event destinations / v2" — the v2 format uses thin payloads and a different signing scheme that this Element does not support.
2. Set the endpoint URL to your deployed base URL plus the webhook path, e.g. `https://your-host/element/stripe/api/stripe/webhook`.
3. Subscribe to the event types you need. Any webhook that arrives with a valid signature is always forwarded as a `RAW_WEBHOOK` event regardless of which types you subscribe to; the typed events below only fire for their corresponding subscribed types.
4. After saving, Stripe shows a **Signing secret** (`whsec_...`). Copy it into the `dev.getelements.elements.stripe.webhook.secret` attribute — see Configuring the Stripe Element.

Testing Webhooks Locally

Forward Stripe events to a local instance with the Stripe CLI:

```
stripe listen --forward-to localhost:8080/element/stripe/api/stripe/webhook
```

Typed Events

Constants live on `StripeEvents` in the `api` module. Each is declared at the class level of the webhook endpoint with `@ElementEventProducer`, and each carries strongly-typed fields pulled straight off the corresponding Stripe object — no payload parsing required in the consumer.

`StripeEvents` constant	Stripe event type	Record
`PAYMENT_SUCCEEDED`	`payment_intent.succeeded`	`StripePaymentSucceededEvent`
`PAYMENT_FAILED`	`payment_intent.payment_failed`	`StripePaymentFailedEvent`

`StripeEvents` constant	Stripe event type	Record
`PAYMENT_CANCELED`	`payment_intent.canceled`	`StripePaymentCanceledEvent`
`INVOICE_PAYMENT_SUCCEEDED`	`invoice.payment_succeeded`	`StripeInvoicePaymentSucceededEvent`
`INVOICE_PAYMENT_FAILED`	`invoice.payment_failed`	`StripeInvoicePaymentFailedEvent`
`SUBSCRIPTION_CREATED`	`customer.subscription.created`	`StripeSubscriptionCreatedEvent`
`SUBSCRIPTION_UPDATED`	`customer.subscription.updated`	`StripeSubscriptionUpdatedEvent`
`SUBSCRIPTION_CANCELLED`	`customer.subscription.deleted`	`StripeSubscriptionCancelledEvent`
`SUBSCRIPTION_TRIAL_WILL_END`	`customer.subscription.trial_will_end`	`StripeSubscriptionTrialWillEndEvent`
`SETUP_INTENT_SUCCEEDED`	`setup_intent.succeeded`	`StripeSetupIntentSucceededEvent`
`PAYMENT_METHOD_ATTACHED`	`payment_method.attached`	`StripePaymentMethodAttachedEvent`

`StripeEvents` constant	Stripe event type	Record
`CHECKOUT_SESSION_COMPLETED`	`checkout.session.completed`	`StripeCheckoutSessionCompletedEvent`
`RAW_WEBHOOK`	`stripe.webhook`	`StripeRawEvent`

The `orgId` field on subscription events is read from the Stripe object's metadata under `StripeService.METADATA_ORG_ID` — it's only populated if the subscription (or its customer) was created with that metadata key set, e.g. via `createCustomer`'s `orgId` parameter or a Checkout Session's `metadata`.

Two of these event types also trigger a side effect inside the webhook handler itself: on `PAYMENT_SUCCEEDED` and `INVOICE_PAYMENT_SUCCEEDED`, the handler calls `StripeService.recordPaymentReceipt` to record a receipt in the platform receipt store, using the payment's `userId` metadata (silently skipped if that metadata is absent or blank).

Consuming Events from Another Element

Any Element that depends on Stripe's `api` module can subscribe to these events with `@ElementEventConsumer`, annotating a method on any Guice-managed service:

```
import com.google.inject.Inject;
import dev.getelements.elements.sdk.annotation.ElementEventConsumer;
import dev.getelements.elements.stripe.StripeEvents;
```

```
public class EntitlementService {

    @Inject
    private UserInventoryDao userInventoryDao;

    @ElementEventConsumer(StripeEvents.PAYMENT_SUCCEEDED)
    public void onPaymentSucceeded() {
        // called whenever a payment_intent.succeeded webhook is received
    }

    @ElementEventConsumer(StripeEvents.SUBSCRIPTION_CANCELLED)
    public void onSubscriptionCancelled() {
        // revoke access, notify player, etc.
    }

    @ElementEventConsumer(StripeEvents.RAW_WEBHOOK)
    public void onAnyWebhook() {
        // called for every verified Stripe webhook — use for event types
        // that don't have a dedicated typed event
    }
}
```

The Webhook Event Log

Every verified webhook — typed or not — is also recorded to a MongoDB-backed audit log (`StripeEventLogService`, collection `stripe_event_log`) storing the Stripe event ID, event type, and receipt timestamp. Browse it via the Stripe tab in the superuser admin panel, or query it directly with `GET /stripe/events?type=&limit=20&offset=0`, which supports filtering by event type and offset-based pagination. This is the fastest way to confirm whether a webhook actually arrived and how it was typed, without digging through application logs.

Related Pages

- Stripe — Element overview and core concepts
- Configuring the Stripe Element — setting the webhook secret and other attributes

- Stripe REST API Reference — the webhook endpoint alongside every other REST endpoint