5. Webhooks

Webhooks let SIR Giving notify your backend when something changes. For example, you can receive an event when an action completes, an action fails, a token pool runs low, or a campaign changes status.

Use webhooks when your system needs a durable server-side record of reward outcomes.

How webhook delivery works

1. You register an HTTPS endpoint.
2. SIR Giving returns a webhook signing secret once.
3. SIR Giving sends events to your endpoint.
4. Your server verifies the signature.
5. Your server stores or queues the event.
6. Your server returns a 2xx response quickly.

Register a webhook

```
POST /v1/partner/webhooks
X-Partner-Key: sk_test_...
X-Timestamp: <timestamp>
X-Signature: <signature>
Content-Type: application/json

{
  "url": "https://your-app.example.com/webhooks/sir",
  "description": "Sandbox webhook",
  "eventTypes": ["action.completed", "action.failed"],
  "receiveAllEvents": false
}
```

Response:

```
{
  "id": "webhook-id",
  "url": "https://your-app.example.com/webhooks/sir",
```

```
  "eventTypes": ["action.completed", "action.failed"],
  "secret": "whsec_...",
  "isActive": true
}
```

Store `secret` immediately. It is shown once.

Delivery headers

Header	Value
`Content-Type`	`application/json`
`User-Agent`	`SIRGiving-Webhooks/1.0`
`X-SIR-Signature`	`sha256=<hex>`
`X-SIR-Timestamp`	Unix timestamp in seconds

Event envelope

```
{
  "id": "evt_abc123",
  "type": "action.completed",
  "createdAt": "2026-05-10T09:12:00Z",
  "data": {
    "actionId": "...",
    "tokensDistributed": 50
  }
}
```

Verify signatures

SIR Giving signs the timestamp and raw request body:

```
signedPayload = X-SIR-Timestamp + "." + rawRequestBody
expected = "sha256=" + HMAC_SHA256_hex(webhookSecret, signedPayload)
```

Compare `expected` to `X-SIR-Signature` using constant-time comparison.

Node.js Express example

```
import crypto from 'crypto';
import express from 'express';

const app = express();

app.use('/webhooks/sir', express.raw({ type: 'application/json' }));

app.post('/webhooks/sir', (req, res) => {
  const signature = req.header('X-SIR-Signature') ?? '';
  const timestamp = req.header('X-SIR-Timestamp') ?? '';
  const rawBody = req.body as Buffer;

  const signedPayload = `${timestamp}.${rawBody.toString()}`;
  const expected = 'sha256=' + crypto
    .createHmac('sha256', process.env.SIR_WEBHOOK_SECRET!)
    .update(signedPayload)
    .digest('hex');

  const valid =
    signature.length === expected.length &&
    crypto.timingSafeEqual(Buffer.from(signature), Buffer.from(expected));

  if (!valid) return res.status(401).end();

  const ageSeconds = Math.abs(Date.now() / 1000 - Number(timestamp));
  if (ageSeconds > 300) return res.status(401).end();

  const event = JSON.parse(rawBody.toString());
  // Store or enqueue the event here.

  return res.status(200).end();
});
```

Retry behavior

- Return a 2xx response when you accept the event.
- Anything else is treated as a failed delivery.
- Failed deliveries are retried with backoff.
- Slow handlers can cause duplicate deliveries, so return quickly and process asynchronously.
- You can inspect delivery history with `GET /v1/partner/webhooks/:id/deliveries`.
- You can manually retry a failed delivery with `POST /v1/partner/webhooks/:id/deliveries/:deliveryId/retry`.

Test your webhook

```
POST /v1/partner/webhooks/:id/test
X-Partner-Key: sk_test_...
X-Timestamp: <timestamp>
X-Signature: <signature>
```

Then check:

```
GET /v1/partner/webhooks/:id/deliveries
```

Common event types

Event	When it fires
`action.completed`	An action processed successfully
`action.failed`	Action processing failed
`action.reversed`	An action was fully or partially reversed
`transaction.completed`	Token distribution transaction completed
`transaction.reversed`	Token distribution transaction reversed
`token_pool.low_balance`	Pool dropped below threshold
`token_pool.depleted`	Pool has no remaining balance
`token_pool.refilled`	Pool was topped up
`token_pool_request.approved`	Allocation request approved
`token_pool_request.rejected`	Allocation request rejected
`campaign.activated`	Campaign moved to active
`campaign.paused`	Campaign paused
`campaign.completed`	Campaign ended

Event	When it fires
`api_key.expiring`	One of your keys is nearing expiry

Revision #4
Created 2026-05-10 09:11:17 UTC by krtin shet
Updated 2026-05-30 13:32:01 UTC by krtin shet