6. End-to-End Scenarios

Use these tutorials when you want to build a real integration path from start to finish.

Scenario 1: Add a donation button

You will create a donation checkout link from the browser using a publishable key.

What you need

- A publishable key, such as `pk_test_...`.
- A nonprofit slug, such as `american-red-cross`.
- A page or component where the user clicks Donate.

Step 1: Create the donation link

```
const response = await fetch('https://devapi.sirgiving.org/v1/partner/donations/create-link',
{
  method: 'POST',
  headers: {
    'X-Partner-Key': 'pk_test_...',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    nonprofitSlug: 'american-red-cross',
    amount: 2500,
    currency: 'USD',
    frequency: 'once',
    provider: 'stripe',
    donorEmail: 'jane@example.com',
    firstName: 'Jane',
    lastName: 'Doe'
  }),
});

const { donationLink, partnerDonationId } = await response.json();
```

```
window.location.href = donationLink;
```

Step 2: Store the donation ID

Store `partnerDonationId` in your session or database. You use it later to show status.

Step 3: Check status

```
GET /v1/partner/donations/status/:partnerDonationId
X-Partner-Key: pk_test_...
```

What you built

You now have a browser-based donation flow. The user clicks your button, SIR Giving creates a checkout link, and you can poll for reward status or listen for webhook events.

Scenario 2: Reward a user for an action

You will submit a backend action when something valuable happens in your system, such as a purchase or completed volunteer shift.

What you need

- A secret key, such as `sk_test_...`.
- The matching HMAC secret.
- An active token pool in the same environment.
- A stable user identifier from your system.

Step 1: Decide your idempotency key

Use a key that maps to one real-world event:

```
purchase_98765
volunteer_shift_abc123
referral_signup_555
```

If you retry the same action, reuse the same key.

Step 2: Submit the action

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

{
  "idempotencyKey": "purchase_98765",
  "actionType": "PURCHASE",
  "amount": 49.99,
  "currency": "USD",
  "stakeholders": [
    {
      "stakeholderTypeCode": "CUSTOMER",
      "partnerUserId": "user_42",
      "userEmail": "customer@example.com",
      "userFirstName": "Jane",
      "userLastName": "Doe"
    }
  ],
  "autoCreateUsers": true,
  "metadata": {
    "orderId": "98765"
  }
}
```

Step 3: Store the action ID

Response:

```
{
  "actionId": "65f1...",
  "idempotencyKey": "purchase_98765",
  "status": "COMPLETED",
  "tokensDistributed": 50,
  "transactionIds": ["65f2..."]
}
```

Store `actionId` with your order. You need it for refunds or support.

What you built

You now have a backend reward flow. Your system sends one signed request, SIR Giving debits your token pool, credits the partner user, and returns a completed result.

Scenario 3: Reverse a reward after a refund

If the real-world action is reversed, reverse the SIR reward too.

```
POST /v1/partner/actions/65f1.../reverse
X-Partner-Key: sk_test_...
X-Timestamp: <timestamp>
X-Signature: <signature>
Content-Type: application/json

{
  "reversalPercentage": 100,
  "reason": "Order refunded",
  "refundIdempotencyKey": "refund_order_98765"
}
```

If the user already spent some rewards, SIR Giving may create a debt that offsets future earnings.

Scenario 4: Register and test a webhook

Step 1: Register the 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",
  "eventTypes": ["action.completed", "action.failed"],
  "receiveAllEvents": false
}
```

Store the returned webhook `secret`.

Step 2: Send a test event

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

Step 3: Inspect delivery history

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

Scenario 5: Preflight before a batch run

Before submitting a large batch:

1. `GET /v1/partner/dashboard/integration-health`
2. `GET /v1/partner/token-pools/:id/balance`
3. `GET /v1/partner/campaigns/active`
4. `GET /v1/partner/dashboard/webhooks/health`
5. `GET /v1/partner/dashboard/api-usage?days=1`

Then submit up to 100 actions at a time:

```
POST /v1/partner/actions/bulk
```

Troubleshooting

Problem	Likely cause	Fix
`INVALID_SIGNATURE`	Signed path/body does not match request	Sign exact path and exact body bytes
`NO_SANDBOX_POOL`	Sandbox key is valid but no sandbox pool exists	Ask SIR Giving to provision a sandbox pool
`No active token pool found`	Production pool missing or inactive	Confirm pool allocation before launch
`403` on write endpoint	You used `pk_...` or missing scope	Use `sk_...` and request needed scope
Duplicate reward concern	Retried request after timeout	Reuse the same `idempotencyKey`
Webhook repeats	Your endpoint did not return 2xx fast enough	Queue work and return quickly

---

Revision #4
Created 2026-05-10 09:11:19 UTC by krtin shet
Updated 2026-05-30 13:32:03 UTC by krtin shet