

Partner API Integration guide

Native coins and USDT across four networks. Check GET `/v1/currencies` for current live availability. Integrate USD-priced invoices, hosted checkout and signed webhooks. All enabled networks remain available in simulated test mode.

API base URL	https://api.coinavio.com
OpenAPI spec	https://api.coinavio.com/v1/partner/openapi.yaml
Latest version	https://api.coinavio.com/v1/partner/docs.md
Contact	support@coinavio.com
Document date	2026-10-25

Reference for payment platforms and shopping carts (panel scripts, billing systems, carts and custom stores) that want to offer **CoinAvio** as a crypto payment method to their merchants.

- Base URL: `https://api.coinavio.com`
- Hosted checkout: `https://coinavio.com/pay/<invoiceId>`
- Machine-readable spec: `https://api.coinavio.com/v1/partner/openapi.yaml`
- Latest copy of this document: `https://api.coinavio.com/v1/partner/docs.md`
- PDF version: `https://coinavio.com/downloads/coinavio-partner-api.pdf`
- Platform integration guide: `https://coinavio.com/docs/platforms`
- Support: support@coinavio.com · <https://coinavio.com/contact>

CoinAvio is a **custodial** gateway. The buyer pays in crypto, the merchant is credited in that coin and withdraws from the CoinAvio dashboard. Your platform never touches wallets, keys or chains; it only creates invoices, redirects the buyer and receives a signed webhook.

1. Concepts

Term	Meaning
Merchant	A CoinAvio account (one per panel/shop owner). Identified by its API key ; no separate merchant id is needed.
API key	<code>ck_live_...</code> or <code>ck_test_...</code> , format <code><keyId>.<secret></code> . Sent as <code>Authorization: Bearer <apiKey></code> . Created by the merchant in <i>Dashboard</i> → <i>Developers</i> .
Invoice	One payment request, priced in USD . The buyer picks the coin on our checkout page.
Webhook secret	<code>whsec_...</code> , shown in <i>Dashboard</i> → <i>Developers</i> . Used to verify webhook signatures. One per merchant.
Test mode	Invoices created with a test key never move crypto and can be marked paid on demand (see §8). Webhooks and statuses behave exactly like live.

Supported currencies

Currency	Networks	Availability
ETH	Ethereum	Read <code>GET /v1/currencies</code> → <code>live</code>
BNB	BNB Smart Chain	Read <code>GET /v1/currencies</code> → <code>live</code>
TRX	TRON	Read <code>GET /v1/currencies</code> → <code>live</code>
TON	TON	Read <code>GET /v1/currencies</code> → <code>live</code>
USDT	Ethereum (ERC20), BNB Smart Chain (BEP20), TRON (TRC20), TON (jetton)	Per-network <code>live: true/false</code> from currencies API

`GET /v1/currencies` and `GET /v1/account` are the authority for live availability. Live checkout offers only live assets; other enabled assets are shown as "Coming soon". Test invoices can simulate every enabled currency, use non-payable placeholder addresses and isolated balances.

Buyer network fee

Ethereum, TRON and TON can add a buyer-paid network fee (BNB Smart Chain has none). It is quoted in the selected coin and locked when the buyer selects it. `pay.merchantAmount` is the coin amount excluding network gas; `pay.networkFee` is `{ amount: "...", currency: "...", usd: "..." }`, and `pay.amount` is the **total to send**. Checkout shows the network fee separately. It is platform gas recovery and is never credited to the merchant or

included in platform percentage fees. Partial payments allocate recovery proportionally, capped at the locked surcharge; ordinary excess-payment credit rules remain in effect. Payment status, tolerance, remaining amount and wallet URI use the total. Webhooks carry the same `pay.networkFee` object. Zero-fee networks and test invoices return zero amounts. The pay block describes the current selection. Previously issued addresses retain their original locked gas quote if a buyer sends to them after changing the payment method; that surcharge still stays out of merchant credit.

2. Payment flow



Credit the order **only from the webhook** (or from `GET /v1/invoices/:id`), never from the buyer landing on `successUrl`.

3. Authentication

```
Authorization: Bearer ck_live_8Fk2...Qp.3f9a...c1
Content-Type: application/json
```

Response	Meaning
401 Invalid API key	Wrong or revoked key.
403 Merchant is not active	The merchant's CoinAvio account is still pending review or was suspended.

Validate a key the merchant pasted into your settings screen with `GET /v1/account` :

```
{
  "merchant": { "id": "cm1...", "name": "Demo Store", "status": "ACTIVE", "feeBps": 0, "feePercent": "0.00" },
  "apiKey": { "test": true, "mode": "test" },
  "site": { "id": "cm9...", "domain": "shop.example.com" },
  "webhook": { "configured": true, "url": "https://panel.example.com/callback/coinavio" },
  "currencies": [ { "currency": "USDT", "network": "BSC", "networkName": "BNB Smart Chain", "live": true,
    "kind": "token", "minAmount": "1", "minConfirmations": 15 }, ... ]
}
```

`site` is `null` for an account-wide key and `{ id, domain }` for a site-scoped key (below).

Site-scoped API keys

In *Dashboard* → *Developers* a merchant can bind a key to **one of their verified sites** (any number of live and test keys per site), or keep the default **account-wide** key. Every key created before 2026-10-24 is account-wide and behaves exactly as before. The key format is the same; a platform does not need to know which kind it was given (`GET /v1/account` tells it).

A site-scoped key:

- **Creates invoices for its site only.** In live mode a `successUrl` or `cancelUrl` that is not on the key's site domain (or a subdomain of it) is replaced with the site root `https://<site domain>/`, keeping nothing from the submitted URL, and reported in `warnings` (see §4). Test keys keep the test-mode exemption.
- **Uses the site's webhook URL** when the site has one (*Dashboard* → *My sites*) and the request has no `callbackUrl`: the URL is stored as the invoice's `callbackUrl` at creation. Changing the site webhook affects new invoices only. An explicit `callbackUrl` always wins. Signing and the webhook secret are unchanged.
- **Sees only its site's invoices.** `GET /v1/invoices`, `GET /v1/invoices/:id`, `GET /v1/invoices/by-order/:orderId`, `cancel` and `simulate` return `404` for invoices of other sites and for invoices created by account-wide keys, the dashboard or payment links. Account-wide keys still see every invoice of the merchant.
- **orderId stays unique per merchant and mode** (not per site). Re-posting an `orderId` of the same site is idempotent as before; if the `orderId` already belongs to another site's or an account-wide invoice the scoped key gets `409 ORDER_ID_IN_USE` (the other invoice is never returned). Prefix order ids per site if several stores share one account.
- When the merchant **removes the site**, its keys are revoked at once (`401 Invalid API key`); invoices are kept.

4. Create an invoice

`POST /v1/invoices`

Field	Type	Required	Notes
<code>priceAmount</code>	string	yes	USD amount, up to 2 decimals, e.g. <code>"49.99"</code> . Minimum <code>0.50</code> , below <code>10¹⁸</code> .
<code>orderId</code>	string ≤128	recommended	Your order/transaction id. Idempotent per merchant and key mode : re-posting the same <code>orderId</code> returns the existing invoice instead of creating a new one. Site-scoped keys: <code>409 ORDER_ID_IN_USE</code> when the <code>orderId</code> belongs to an invoice outside the key's site.

Field	Type	Required	Notes
<code>description</code>	string ≤500	no	Shown to the buyer on the checkout page.
<code>successUrl</code>	string ≤2048	no	Where the buyer is sent after paying. Any spelling accepted (<code>shop.com/thanks</code> becomes <code>https://shop.com/thanks</code>). Live keys: a host on a domain the merchant verified in <i>Dashboard</i> → <i>My sites</i> (its <code>www.</code> host and subdomains count) is used as given; any other host is replaced, not rejected (see <i>Return URLs and warnings</i> below). An address with another scheme, an IP address instead of a domain name, or one that cannot be parsed returns <code>400</code> with a readable <code>message</code> . Test keys: no domain check.
<code>cancelUrl</code>	string ≤2048	no	"Back to shop" link on the checkout page. Same rules as <code>successUrl</code> .
<code>callbackUrl</code>	string ≤2048	no	Webhook URL for this invoice only . Overrides the merchant's account-level webhook URL. Must be HTTPS on a public host (ports 443, 8443, 80 or 8080), without credentials in the URL. The host is re-validated at every delivery. Use this when your platform routes callbacks per order. Site-scoped keys: when omitted, the site's webhook URL (if set) is stored here.
<code>buyerPaysFee</code>	boolean	no	<code>true</code> adds the CoinAvio fee on top so the merchant receives the full <code>priceAmount</code> . The buyer is charged <code>price.amount</code> , your requested amount is <code>price.baseAmount</code> . With a 0 % fee both are equal.
<code>metadata</code>	object	no	Free-form JSON, returned unchanged on the invoice and in webhooks.
<code>expiresInMinutes</code>	integer 5–1440	no	Payment window from creation. Default 60 minutes . Integer values are clamped to 5-1440; fractional or nonnumeric values are rejected. Selecting or switching a coin never extends or shortens the deadline; the selected crypto amount is fixed until expiry.

```
curl -X POST https://api.coinavio.com/v1/invoices \
-H "Authorization: Bearer $COINAVIO_API_KEY" \
-H "Content-Type: application/json" \
-d '{
  "priceAmount": "25.00",
  "orderId": "shop-17:1048",
  "description": "Balance top-up - panel.example.com",
  "successUrl": "https://panel.example.com/addfunds?status=success",
  "cancelUrl": "https://panel.example.com/addfunds",
  "callbackUrl": "https://panel.example.com/callback/coinavio",
  "buyerPaysFee": true,
  "metadata": { "userId": 8812 }
}'
```

Response `201` :

```
{
  "id": "cmgq1x8k0000a...",
  "orderId": "shop-17:1048",
  "status": "NEw",
  "testMode": false,
  "description": "Balance top-up - panel.example.com",
  "price": { "amount": "25", "currency": "USD", "baseAmount": "25", "buyerPaysFee": false, "feeBps": 0,
"fee": "0" },
  "pay": null,
  "paidAmount": "0",
  "paidUsd": "0",
  "checkoutUrl": "https://coinavio.com/pay/cmgq1x8k0000a...",
  "expiresAt": "2026-10-13T10:45:00.000Z",
  "paymentDeadline": "2026-10-13T10:45:00.000Z",
  "underpaidAccepted": false,
  "underpaidAcceptedAt": null,
  "underpaidAcceptedBy": null,
  "paidAt": null,
  "createdAt": "2026-10-13T09:45:00.000Z",
  "successUrl": "https://panel.example.com/addfunds?status=success",
  "cancelUrl": "https://panel.example.com/addfunds",
  "callbackUrl": "https://panel.example.com/callback/coinavio",
  "metadata": { "userId": 8812 },
  "payments": [],
  "warnings": []
}
```

Amounts are decimal strings without trailing zeros ("25" , "49.9"). Compare them as numbers, not as strings.

Redirect the buyer to `checkoutUrl`. Once the buyer selects a coin, `pay` is filled:

```
"pay": {
  "amount": "25", "received": "0", "shortfall": "25", "shortfallPercent": "100.00", "tolerancePercent":
"0.00", "remainingAmount": "25", "currency": "USDT", "network": "BSC", "networkName": "BNB Smart Chain",
  "address": "0x111111111111111111111111111111111111", "memo": null,
  "uri": "ethereum:0x55d398326f99059f775485246999027B3197955@56/transfer?
address=0x111111111111111111111111111111111111&uint256=25000000000000000000",
  "rate": "1", "rateLockedAt": "...", "minConfirmations": 15,
  "contractAddress": "0x55d398326f99059f775485246999027B3197955", "explorerAddressUrl":
"https://bscscan.com/address/0x111111111111111111111111111111111111"
}
```

`pay.remainingAmount` is the full quote before any receipt. After a partial receipt, it is the base-unit amount still needed to reach the paid threshold, floored to the asset's decimals. `pay.shortfall` remains the full expected minus received amount. While an invoice is partially paid, `pay.uri` asks only for the remainder. EVM checkout QR codes encode the EIP-681 URI with network and exact base-unit amount; TON URIs include the exact memo. Use the displayed address and amount as a fallback if your wallet cannot parse the URI. Test invoices use non-payable placeholder addresses, an empty `pay.uri` and a null `pay.explorerAddressUrl`.

For live TRON, its URI formatter uses the plain address: `{"address": "TR7NHq...", "memo": null, "uri": "TR7NHq..."}`. This is a formatting example, not a payable test target.

Return URLs and warnings

Live invoices only (test invoices have no domain check):

- A `successUrl` or `cancelUrl` whose host is on one of the merchant's verified domains (including its `www.` host and subdomains) is used as given.
- **Account-wide key**, any other host: the URL is replaced with the merchant's verified website URL, or dropped (no return link on the checkout page) when no website URL is verified.
- **Site-scoped key**, a host outside the key's site: the URL is replaced with the site root `https://<site domain>/`. Nothing from the submitted URL is kept.
- An address that is not a valid web address (another scheme, an IP address instead of a domain name, or text that cannot be parsed) still returns `400`, in every mode, as a validation error with a readable `message`.
- Payment links created in the dashboard still require verified domains (`DOMAIN_NOT_VERIFIED` applies only there).

`POST /v1/invoices` responses always include `warnings`, an array that is empty when nothing was changed, in test mode and on an idempotent `orderId` replay. Each replaced field adds one entry:

```
"warnings": [  
  { "code": "RETURN_URL_REPLACED", "field": "successUrl", "message": "..." }  
]
```

`field` is `successUrl` or `cancelUrl`. `message` explains the change for people (for example, which host was not verified and where the buyer will return instead); its wording may change, so do not parse it. `warnings` is not stored and is not part of `GET`, list or cancel responses or webhooks. The invoice's `successUrl` and `cancelUrl` in the response show the final stored values (`null` when a URL was dropped).

Invoice statuses

Status	Meaning	Final
NEW	Created, buyer has not paid yet (may or may not have picked a coin).	
PENDING	Transaction seen on chain, waiting for confirmations.	
CONFIRMING	Partially confirmed / paid, or verification continuing while closure is held.	
PAID	Amount confirmed in full, within tolerance, or explicitly accepted. Fulfil an eligible store order once; preserve protected store states for review.	✓
OVERPAID	More than the amount confirmed. Fulfil the order; the surplus is credited to the merchant.	✓
UNDERPAID	Window closed with a partial payment. The partial amount is credited to the merchant; the order is not paid.	✓
EXPIRED	Payment window closed; later money remains credited and reviewable without reopening.	✓
CANCELLED	Cancelled through the API while still <code>NEW</code> .	✓

Compliance screening. A received payment may be screened before it is credited. While screening or under review the invoice keeps its current unpaid status (`PENDING` / `CONFIRMING`) and does not expire, and `reviewReason` is `AML_SCREENING` or `AML_REVIEW`; `invoice.paid` (and the final `PAID` / `OVERPAID` status) is sent only after release; earlier `invoice.pending` / `invoice.confirming` events behave as before. If the payment is rejected, `reviewReason` becomes `AML_REJECTED` and the invoice expires normally; contact support. `AML_REJECTED` can also appear on an

invoice that was paid by another receipt. Each entry in `payments` carries a boolean `held` (`true` while that receipt is being screened or reviewed); its `netCredit` stays `null` until the receipt is released. There are no new invoice statuses or webhook event types. Do not fulfil orders from `reviewReason` values.

An invoice counts as paid when confirmed selected-coin receipts reach $\text{pay.amount} * (100 - \text{pay.tolerancePercent}) / 100$, computed with decimal arithmetic and rounded DOWN to the asset's decimals, then compared in integer base units. Two-decimal percentages below 1 %, including 0.01 %, 0.1 %, 0.25 % and 0.99 %, work exactly. After a partial receipt, `pay.remainingAmount` and the wallet URI request only the amount needed to reach this same paid threshold; before any receipt they request the full quote. `pay.shortfall` always shows the full expected minus received amount, even when the remaining top-up is zero. Test-mode simulation uses the same threshold.

Merchants choose their underpayment tolerance in *Dashboard* → *Settings* (0 through the administrator maximum, up to 2 decimals). A null merchant preference uses the platform default, initially **0 % - exact amount required**; the initial maximum is **10 %**, which the administrator may raise up to 100 %. New invoices snapshot the effective tolerance at creation. Lowering the maximum caps preferences on new invoices; existing invoices retain their terms. Invoices without a tolerance snapshot also use 0 %; already-paid invoices remain paid.

Tolerance applies to the **total including any buyer-paid network fee**. Merchant credit is the actual received principal, less the existing percentage fee on received principal and buyer-paid network recovery; the shortfall is never credited. Overpayment classification retains its existing 0.5 % excess threshold. A PENDING/CONFIRMING invoice has a 60-minute grace period after `expiresAt` for confirmations and top-ups. `paymentDeadline` exposes the effective deadline; the original locked amount remains unchanged during grace. Every positive receipt at an issued invoice address is recorded, including a first partial payment or a top-up below `minDeposit`. The minimum still applies when selecting the invoice's full payment amount.

Each entry in `payments` has a unique `id`, its actual `currency` and `network`, `requiredConfirmations`, `matchesSelectedAsset`, `fee` and `networkFeeRecovered`. A payment in a different supported coin is credited to the merchant in that coin but does not settle the invoice; unsupported tokens and other networks must not be sent. TON payments must include the exact plain-text memo shown at checkout. A TON payment with a missing, different, encrypted or binary comment is not matched automatically: it is held for manual review (encrypted comments are not decrypted) and never credited twice. Test payments have `explorerUrl: null`.

EXPIRED and CANCELLED never reopen. UNDERPAID never reopens automatically from a late receipt; only the merchant's explicit acceptance in the dashboard (below) can change it to PAID. Late receipts still credit the merchant balance and emit `invoice.payment_confirmed`, with the final invoice status preserved. Previously issued addresses remain attributed after a coin switch; payments in a different asset credit that asset balance but do not count toward the selected-asset invoice total. PAID can become OVERPAID after additional receipts.

`paidUsd` is the USD value of confirmed payments in the selected asset at confirmation time; a price outage leaves that payment's `usdValue` null without preventing crypto credit. Each webhook snapshot is recorded in the transaction that decides the transition.

Test keys can read, list, look up and cancel only test invoices; live keys can access only live invoices. A mode mismatch returns 404. The same `orderId` can exist once in each mode. Dashboard sessions see both modes.

5. Read, look up, cancel

Endpoint	Purpose
<code>GET /v1/invoices/:id</code>	Full invoice (same shape as above). Safe to poll every few seconds if you cannot receive webhooks.

Endpoint	Purpose
GET /v1/invoices/by-order/:orderId	Same, by your own <code>orderId</code> .
GET /v1/invoices?status=PAID&limit=50&cursor=...	Paginated list, newest first. <code>nextCursor</code> is <code>null</code> on the last page.
POST /v1/invoices/:id/cancel	Owner merchant key and matching mode (site-scoped keys: own site only). Cancels NEW only when no receipt exists; CANCELLED retries are idempotent. A receipt refuses cancellation with 400 and records STORE_CANCELLED_WITH_RECEIPT review.

Accept an underpayment

Accepting an underpayment is **dashboard-only**: the merchant opens the invoice in *Dashboard* → *Invoices* and chooses **Accept as paid**. An API key cannot accept an underpayment. `POST /v1/invoices/:id/accept-underpayment` is still routed, but with any valid key and for any invoice it returns `403` and never changes the invoice:

```
{ "error": "ACCEPT_UNDERPAYMENT_DASHBOARD_ONLY", "message": "Accept underpayments in the CoinAvio dashboard (Invoices → Accept as paid)." }
```

Platforms should keep the order on hold and wait for `invoice.paid` (or poll the invoice).

The sole transition is **UNDERPAID -> PAID**. Sets `underpaidAccepted: true`, `underpaidAcceptedAt`, `underpaidAcceptedBy` (merchant user or administrator; invoices accepted through the API before 2026-10-25 show the API-key identity), and `paidAt`; emits an immutable `invoice.paid` snapshot. Funds were credited on confirmation, so acceptance makes **no ledger movement**. Accepting an already accepted invoice again sends no second event. Invoices in any other status cannot be accepted.

For both paid and underpaid webhook snapshots, `pay.amount` is expected, `pay.received` is confirmed selected-coin receipts, `pay.shortfall` is $\max(\text{expected} - \text{received}, 0)$, `pay.shortfallPercent` is the missing percentage (2 decimals), and `pay.tolerancePercent` is the invoice's tolerance. `underpaidAccepted` is true for an automatically accepted shortfall or explicit acceptance; the acceptance actor/time are populated only for the explicit action. For example, if the merchant explicitly chooses 0.5 % tolerance, a 20-USDT invoice paid with 19.92 is PAID with received 19.92, shortfall 0.08 and `underpaidAccepted` true. A receipt of 15 is UNDERPAID after the existing grace deadline, showing shortfall 5 (25 %).

6. Webhooks

CoinAvio POSTs JSON to the invoice's `callbackUrl`, or, when none was given, to the merchant's account-level webhook URL (*Dashboard* → *Developers*). If neither is set, no webhook is sent. Invoices of a site-scoped key without their own `callbackUrl` get the site's webhook URL (*Dashboard* → *My sites*) as `callbackUrl` when they are created. All deliveries are signed with the same account webhook secret.

Headers:

```
Content-Type: application/json
User-Agent: Coinavio-Webhooks/1.0
X-Webhook-Id: <event id, unique>
X-Webhook-Event: invoice.paid
X-Webhook-Timestamp: 1760348700000 (unix ms)
X-Webhook-Signature: t=1760348700000,v1=<hex>
X-CoinAvio-Mode: live
X-CoinAvio-Mode-Signature: t=1760348700000,v1=<mode-bound hex>
```

Body:

```
{
  "id": "evt_cmqq...",
  "type": "invoice.paid",
  "createdAt": "2026-10-13T09:52:10.000Z",
  "data": { "invoice": { ...same object as GET /v1/invoices/:id... } }
}
```

Event types:

Type	When
invoice.pending	First transaction detected.
invoice.payment_detected	Additional, different-method or late receipt detected; inspect the payment records and overall invoice status.
invoice.payment_orphaned	A detected receipt is proven failed after finality. Informational; it does not authorize order credit or reverse a previously confirmed credit.
invoice.confirming	Partial confirmation / partial payment.
invoice.paid	Paid in full, within tolerance, or explicitly accepted. Complete the order here. If <code>underpaidAccepted</code> is true, the snapshot shows the accepted shortfall.
invoice.overpaid	Paid more than requested. Treat like <code>paid</code> .
invoice.underpaid	Closed with a partial payment. Hold the order for review; expected / received / missing are in <code>pay.amount</code> , <code>pay.received</code> , <code>pay.shortfall</code> and <code>pay.shortfallPercent</code> . The merchant balance already receives the actual funds.
invoice.expired	Closed with no payment.
invoice.cancelled	Cancelled while NEW.
invoice.payment_confirmed	An additional transaction was confirmed without changing the status (e.g. a second transfer). Informational.

Type	When
<code>invoice.review_requested</code>	A store cancellation was refused because the invoice already has a receipt. Informational.
<code>invoice.discovery_delayed</code>	Network discovery/finality needs review after 24 hours past the effective deadline. Informational; do not fulfill or request another payment.
<code>withdrawal.completed</code> / <code>withdrawal.failed</code> / <code>withdrawal.rejected</code>	Merchant withdrawals; account-level webhook only.

Delivery rules:

- Respond with any `2xx` within 10 seconds; keep the body within 4 KB. Successful oversized responses are truncated to 4 KB and recorded as delivered. Anything else is retried with exponential backoff, 8 attempts: initial delivery, then delays of 30, 60, 120, 240, 480, 960 and 1920 seconds (63.5 minutes total delay, plus request durations). Responses are capped at 4 KB while streaming.
- Deliveries can arrive **more than once** and **out of order**. Deduplicate on `id` (or `X-Webhook-Id`) and treat `data.invoice.status` as the source of truth.
- After the eighth failed attempt the event is marked exhausted. The merchant can redeliver it from *Dashboard* → *Developers* (Redeliver); redelivery sends the same event snapshot.
- Redirects are not followed. Use the final HTTPS URL.

Informational events. `invoice.payment_detected`, `invoice.payment_confirmed`, `invoice.payment_orphaned`, `invoice.review_requested` and `invoice.discovery_delayed` report receipts and reviews. They never authorize crediting an order. Do not fulfil orders from `reviewReason` values either.

Wrong coin, late or short payments. A payment in another coin, or one that arrives after the invoice closed, is credited to the merchant's CoinAvio balance but does not complete the order. CoinAvio does not refund automatically: the buyer contacts the merchant (<https://coinavio.com/refunds>, <https://coinavio.com/help#payment-problems>).

Signature verification

`v1 = HMAC_SHA256(webhookSecret, "<t>.<raw request body>")` in lowercase hex. Reject if `|now - t| > 5 minutes` or the comparison fails. Always hash the **raw** body bytes, not a re-serialised object.

The original signature and merchant secret stay unchanged. Two additive headers bind the mode to the same raw body: `X-CoinAvio-Mode: test|live` and `X-CoinAvio-Mode-Signature: t=<ms>,v1=<hex>`. The second HMAC input is `<t>.<mode>.<raw request body>` using the same secret and timestamp as the original signature. Verify both signatures and equal timestamps. Never trust the mode header without its signature.

The following examples are **production invoice handlers**: reject a mode other than `live`, reject `testMode:true`, and require `data.invoice.testMode === false` (missing mode also fails closed). Use a separate test workflow for simulated payments. Withdrawal events need a separate handler with the same signature/mode checks; they never fulfil orders. After verification, match invoice ID, order ID, currency and amount, and process only PAID/OVERPAID for fulfilment with an atomic event-ID deduplication record.

PHP:

```

<?php
// Production invoice handler. Route withdrawal events separately.
function verify_coinavio_webhook($secret, $signature, $rawBody, $mode, $modeSignature) {
    if (!$secret || $mode !== 'live') return false;
    $pattern = '/^t=([0-9]+),v1=([a-zA-F0-9]{64})$/D';
    if (!is_string($signature) || !is_string($modeSignature) ||
        !preg_match($pattern, $signature, $a) ||
        !preg_match($pattern, $modeSignature, $b) || $a[1] !== $b[1]) return false;
    if (abs(microtime(true) * 1000 - (float) $a[1]) > 300000) return false;
    $legacy = hash_hmac('sha256', $a[1] . '.' . $rawBody, $secret);
    $bound = hash_hmac('sha256', $a[1] . '.' . $mode . '.' . $rawBody, $secret);
    if (!hash_equals($legacy, strtolower($a[2])) ||
        !hash_equals($bound, strtolower($b[2]))) return false;
    $event = json_decode($rawBody, true);
    return is_array($event) && is_string($event['type'] ?? null) &&
        strpos($event['type'], 'invoice.') === 0 &&
        ($event['data']['invoice']['testMode'] ?? null) === false;
}

// In your HTTP handler, before processing an order:
$rawBody = file_get_contents('php://input');
if ($rawBody === false || !verify_coinavio_webhook(
    getenv('COINAVIO_WEBHOOK_SECRET'), $_SERVER['HTTP_X_WEBHOOK_SIGNATURE'] ?? '',
    $rawBody, $_SERVER['HTTP_X_COINAVIO_MODE'] ?? '',
    $_SERVER['HTTP_X_COINAVIO_MODE_SIGNATURE'] ?? ''
)) { http_response_code(400); exit; }
$event = json_decode($rawBody, true, 512, JSON_THROW_ON_ERROR);
// Match invoice/order, currency and amount; only PAID/OVERPAID may fulfil.
// Persist event ID and order update in one transaction before returning 200.
http_response_code(200);

```

Node.js:

```

import { createHmac, timingSafeEqual } from 'node:crypto';

// Production invoice handler. Route withdrawal events separately.
export function verifyWebhook(secret, signature, rawBody, mode, modeSignature) {
  if (!secret || mode !== 'live') return false;
  const pattern = /^t=([0-9]+),v1=([a-zA-F0-9]{64})$/;
  const legacy = typeof signature === 'string' && pattern.exec(signature);
  const signedMode = typeof modeSignature === 'string' && pattern.exec(modeSignature);
  if (!legacy || !signedMode || legacy[1] !== signedMode[1]) return false;
  const timestampMs = Number(legacy[1]);
  if (!Number.isSafeInteger(timestampMs) ||
    Math.abs(Date.now() - timestampMs) > 300_000) return false;
  const valid = (prefix, hex) => {
    const expected = createHmac('sha256', secret).update(prefix).update(rawBody).digest();
    return timingSafeEqual(expected, Buffer.from(hex, 'hex'));
  };
  if (!valid(legacy[1] + '.', legacy[2]) ||
    !valid(legacy[1] + '.' + mode + '.', signedMode[2])) return false;
  try {
    const event = JSON.parse(rawBody.toString('utf8'));
    return typeof event.type === 'string' && event.type.startsWith('invoice.') &&
      event.data?.invoice?.testMode === false;
  } catch { return false; }
}

// Read the exact raw bytes BEFORE JSON middleware:
// const rawBody = Buffer.concat(await Array.fromAsync(req));
// if (!verifyWebhook(process.env.COINAVIO_WEBHOOK_SECRET,
//   req.headers['x-webhook-signature'], rawBody,
//   req.headers['x-coinavio-mode'], req.headers['x-coinavio-mode-signature'])) {
//   res.writeHead(400); res.end(); return;
// }
// const event = JSON.parse(rawBody.toString('utf8'));
// Match invoice/order, currency and amount; only PAID/OVERPAID may fulfil.
// Persist event.id and the order update in one transaction before returning 200.

```

Python:

```

import hashlib, hmac, json, re, time

# Production invoice handler. Route withdrawal events separately.
def verify_coinavio(secret, signature, raw_body, mode, mode_signature):
    if not secret or mode != 'live':
        return False
    pattern = r't=([0-9]+),v1=([a-zA-F0-9]{64})'
    a = re.fullmatch(pattern, signature or '')
    b = re.fullmatch(pattern, mode_signature or '')
    if not a or not b or a[1] != b[1]:
        return False
    if abs(time.time() * 1000 - int(a[1])) > 300_000:
        return False
    def valid(prefix, digest):
        expected = hmac.new(secret.encode(), prefix.encode() + raw_body,
                             hashlib.sha256).hexdigest()
        return hmac.compare_digest(expected, digest.lower())
    if not valid(a[1] + '.', a[2]) or not valid(a[1] + '.' + mode + '.', b[2]):
        return False
    try:
        event = json.loads(raw_body)
        return event['type'].startswith('invoice.') and \
            event['data']['invoice']['testMode'] is False
    except (ValueError, KeyError, TypeError, AttributeError):
        return False

# Reject with HTTP 400 when verify_coinavio(...) is False.
# Match invoice/order, currency and amount; only PAID/OVERPAID may fulfil.
# Persist event ID and order update in one transaction before returning 200.

```

Withdrawal events use `data.withdrawal`, not `data.invoice`. The withdrawal object includes id, status, currency, network, toAddress, memo, addressLabel, amount, fee, usdValue, txHash, explorerUrl, failureReason, manualApprovalReason, merchant, approvedBy, approvedAt, emailConfirmedAt, createdAt and updatedAt. `manualApprovalReason` is a nullable explanation of why automatic approval was not allowed; integrations must allow additive fields. Manual resolution emits completed/failed; rejection and email expiry emit rejected.

7. Fees and "buyer pays fee"

The CoinAvio fee is a percentage per merchant (`feeBps` , 100 bps = 1 %), currently **0 % for the first six months**. Each invoice snapshots `feeBps` at creation; later merchant fee changes do not affect its receipts. It is deducted from the crypto credited to the merchant when a payment confirms.

With `buyerPaysFee: true` the invoice is grossed up: `price.amount = ceil2(priceAmount / (1 - feeBps/10000))`, so the merchant nets at least `priceAmount` at the locked quote rate (crypto is credited; confirmation-time USD valuation can differ). `price.fee` is the USD fee for display. Credit the order for `price.baseAmount` .

Each payment exposes `fee` and `networkFeeRecovered` in its currency. Merchant credit is amount minus these fields. Percentage fees use cumulative principal per invoice/asset, preventing rounding drift when a buyer splits payments.

8. Test mode

1. After approval, the merchant creates a **test key** (`ck_test_...`) in *Dashboard* → *Developers*.
2. Invoices created with it have `testMode: true`; the checkout page shows a *Test mode* badge and a **Simulate payment** button once a coin is selected.
3. From your server you can do the same: `POST /v1/invoices/:id/simulate` (test key, coin selected) → the invoice becomes `PAID`, usually within about two minutes, and the normal `invoice.paid` webhook fires. After a partial payment, only the remaining amount is simulated.
4. Return-URL domain verification is **not** enforced for test invoices.
5. Test and live keys use separate invoice namespaces: a test key never sees live invoices, and the same `orderId` can exist once in each mode.
6. Simulation works on every enabled network. Test addresses are non-payable placeholders with no wallet URI or explorer link. **Do not send real funds to a test invoice.** Test balances are isolated from live balances and can never be withdrawn.

9. Errors

Domain errors can use `{ "error": "ORDER_ID_IN_USE", "message": "..." }` without `statusCode`. Framework validation errors use the shape below. All operations can return 429; key-auth operations can return 401/403. Unknown `CreateInvoice` fields are rejected; optional fields accept null as omitted.

```
{ "statusCode": 400, "message": "priceAmount must be a USD amount with up to 2 decimals, e.g. \"49.99\"", "error": "Bad Request" }
```

Status	Typical causes
400	Validation (see <code>message</code> , may be an array), including a return or callback URL that is not a valid web address; cancelling a non- <code>NEW</code> invoice; simulate on a live invoice.
409	<code>ORDER_ID_IN_USE</code> : a site-scoped key reused an <code>orderId</code> that belongs to another site's or an account-wide invoice.
401	Missing/invalid API key.
403	Merchant account not <code>ACTIVE</code> ; <code>ACCEPT_UNDERPAYMENT_DASHBOARD_ONLY</code> from <code>POST /v1/invoices/:id/accept-underpayment</code> (accepting an underpayment is dashboard-only).
404	Unknown invoice / <code>orderId</code> (or it belongs to another merchant, or — for a site-scoped key — to another site).
429	Rate limit (300 requests per minute per IP; invoice creation 60/min/IP). See <i>Limits</i> below.
413	Request body exceeds 64 KiB.

Limits

Item	Limit
Minimum invoice	0.50 USD, checked on the amount the buyer is charged (after any buyer-paid fee)
<code>priceAmount</code>	JSON string of digits with up to 2 decimals (<code>"49.99"</code> , <code>"25"</code>), at most 21 characters, below 10^{18}
<code>orderId</code>	String up to 128 characters, no character restrictions. Idempotent per merchant and mode. URL-encode it in <code>GET /v1/invoices/by-order/:orderId</code>

Item	Limit
<code>description</code>	Up to 500 characters
<code>successUrl</code> , <code>cancelUrl</code> , <code>callbackUrl</code>	Up to 2048 characters each
Payment window	60 minutes by default; <code>expiresInMinutes</code> 5–1440. The crypto amount is fixed from coin selection until expiry
Grace period	60 minutes after <code>expiresAt</code> for <code>PENDING</code> / <code>CONFIRMING</code> invoices (<code>paymentDeadline</code>), for confirmations and top-ups
Rate limits	300 requests per minute per IP across the API; <code>POST /v1/invoices</code> 60 per minute per IP. Excess returns <code>429</code>
Request body	64 KiB
List page size	<code>limit</code> 1–100, default 20
Webhook response	<code>2xx</code> within 10 seconds; only the first 4 KB of the response is read
Webhook retries	8 attempts: the first delivery, then waits of 30 s, 1, 2, 4, 8, 16 and 32 minutes (63.5 minutes of waits in total)
Signature timestamp	Reject when <code>t</code> differs from your clock by more than 5 minutes

10. Mapping from other gateways

If your platform already has a Cryptomus/NOWPayments-style module, the concepts map 1:1:

Cryptomus	CoinAvio
<code>merchant</code> UUID + <code>API key</code> + <code>sign</code> (MD5) header	Single <code>Authorization: Bearer <apiKey></code> header, no request signing
<code>POST /v1/payment</code> <code>amount</code> , <code>currency: "USD"</code>	<code>POST /v1/invoices</code> <code>priceAmount</code> (always USD)
<code>order_id</code>	<code>orderId</code> (idempotent)
<code>url_return</code> , <code>url_success</code> , <code>url_callback</code>	<code>cancelUrl</code> , <code>successUrl</code> , <code>callbackUrl</code>
<code>subtract</code> (who pays the commission)	<code>buyerPaysFee: true/false</code>
<code>additional_data</code>	<code>metadata</code> (JSON)
<code>result.url</code>	<code>checkoutUrl</code>
<code>result.status</code> <code>paid</code> / <code>paid_over</code> / <code>wrong_amount</code> / <code>cancel</code>	<code>PAID</code> / <code>OVERPAID</code> / <code>UNDERPAID</code> / <code>EXPIRED</code>
Webhook <code>sign</code> (MD5 of body + key)	<code>X-Webhook-Signature: t=...,v1=HMAC-SHA256</code> (see §6)
<code>GET /v1/payment/info</code>	<code>GET /v1/invoices/:id</code> or <code>/v1/invoices/by-order/:orderId</code>
Test mode	<code>ck_test_</code> keys + <code>/simulate</code>

11. Integration checklist for platforms

Full walkthrough with a complete PHP example: <https://coinavio.com/docs/platforms>

- ☐ Settings screen: *API key* (required; `ck_live_...` or `ck_test_...`, detect the mode from the prefix) and *Webhook secret* (required; `whsec_...`, the **Signing secret** in *Dashboard* → *Developers*). Optional *Buyer pays fee* switch. No merchant id. Optional: call `GET /v1/account` on save and show the merchant name and mode.
- ☐ Create invoices with `priceAmount` (USD string), `orderId` = `<site-id>:<payment-id>`, `successUrl`, `cancelUrl`, `callbackUrl` (your own endpoint, per invoice, with your site or tenant id in the path) and `buyerPaysFee` from the setting.
- ☐ Save the invoice `id`, log any `warnings`, and redirect to `checkoutUrl` (HTTP 303).
- ☐ Webhook endpoint: verify both signatures and the timestamp on the raw body, deduplicate on the event `id`, credit `price.baseAmount` only on `invoice.paid` / `invoice.overpaid`, hold `invoice.underpaid`, ignore informational events. Return `2xx` only after your update is committed.
- ☐ Optional fallback: read `GET /v1/invoices/by-order/:orderId` for payments still pending after the buyer returns, and apply the same rules.
- ☐ Tested end-to-end with a `ck_test_` key and `/simulate`.

Status mapping

CoinAvio	Common platform state	What to do
<code>PAID</code> , <code>OVERPAID</code>	Completed	Credit <code>price.baseAmount</code> once, on <code>invoice.paid</code> / <code>invoice.overpaid</code> .
<code>UNDERPAID</code>	Underpaid / on hold	Do not credit automatically. If the merchant accepts the shortfall, <code>invoice.paid</code> follows and the status becomes <code>PAID</code> .
<code>NEW</code>	Awaiting payment	Nothing to do.
<code>PENDING</code> , <code>CONFIRMING</code>	Pending	Wait for the next webhook.
<code>reviewReason</code> <code>AML_SCREENING</code> or <code>AML_REVIEW</code>	On hold (pending)	Compliance review of a received payment. The invoice stays <code>PENDING</code> / <code>CONFIRMING</code> and does not expire; <code>invoice.paid</code> arrives after release.
<code>EXPIRED</code> , <code>CANCELLED</code>	Expired / failed	Do not credit.

Treat the status in the newest event as the source of truth and never let an older event undo a completed payment.

Amount to credit

Credit `price.baseAmount` (the USD amount you requested), only on `invoice.paid` or `invoice.overpaid` with status `PAID` or `OVERPAID`. Do not use `price.amount` (includes a buyer-paid fee), `pay.amount` (crypto) or `paidUsd` (valuation at confirmation). Amounts have no trailing zeros ("25"); compare them as numbers. On an overpayment, credit the requested amount; the surplus stays in the merchant's CoinAvio balance.

Several sites per merchant

`orderId` is idempotent per merchant and mode: re-posting it returns the existing invoice. Use `<site-id>:<payment-id>` so two sites of one merchant never collide. Deduplicate webhooks on the event `id`, and check that the invoice `id` and `orderId` belong to the merchant whose secret verified the webhook.

Fees and currency

If your platform already charges the buyer a fee for this payment method, keep `buyerPaysFee` off so the buyer is not charged twice. Invoices are always USD: convert from the merchant's currency before creating the invoice and credit the stored local amount when it is paid.

Return URLs on extra domains

Merchants often serve one store on extra domains (mirrors) they never verified. For live invoices such a `successUrl` or `cancelUrl` does not fail: it is replaced with the merchant's verified website URL (account-wide key) or the key's site root (site-scoped key), or dropped when the merchant has no verified website, and the create response lists the change in `warnings` (see §4). To send buyers back to the exact page, use a domain the merchant verified in *Dashboard* → *My sites*. `callbackUrl` needs no verification.

12. Changelog

Date	Change
2026-10-25	Removing a verified site in <i>Dashboard</i> → <i>My sites</i> (which revokes its site-scoped keys) now asks for a fresh authenticator code, or the account password when 2FA is off; unverified sites without keys need only a confirmation. <i>My sites</i> shows each site's icon. API-key requests are unchanged.
2026-10-25	Accepting an underpayment is now dashboard-only; the API-key route returns 403 <code>ACCEPT_UNDERPAYMENT_DASHBOARD_ONLY</code> .
2026-10-24	Return URLs on unverified domains no longer fail live invoice creation; they are replaced (merchant website or the key's site root) and reported in a new <code>warnings</code> array on create responses.
2026-10-24	Platform integration guide published (https://coinavio.com/docs/platforms). Docs reorganised: Limits section, status mapping for platforms, partner-facing changelog; internal notes removed.
2026-10-24	Site-scoped API keys: a key can be bound to one verified site. Scoped keys: live return URLs must be on that site, invoices get the site's webhook URL as <code>callbackUrl</code> when none is sent, reads are limited to the site's invoices (others 404), and reusing another site's <code>orderId</code> returns 409 <code>ORDER_ID_IN_USE</code> . <code>GET /v1/account</code> adds <code>site</code> (<code>null</code> for account-wide keys). Removing a site revokes its keys. Existing keys are account-wide and unchanged; invoice and webhook JSON unchanged.
2026-10-23	Compliance screening: <code>reviewReason</code> values <code>AML_SCREENING</code> , <code>AML_REVIEW</code> , <code>AML_REJECTED</code> ; payments gain <code>held</code> ; <code>invoice.paid</code> is sent only after release.
2026-10-22	Signed mode headers <code>X-CoinAvio-Mode</code> and <code>X-CoinAvio-Mode-Signature</code> . The original signature and the webhook secret are unchanged.
2026-10-22	Buyer network fee on TRON and TON as well as Ethereum; <code>pay.networkFee</code> shape unchanged.
2026-10-22	<code>invoice.discovery_delayed</code> and <code>reviewReason</code> <code>PAYMENT_DISCOVERY_DELAYED</code> (informational).
2026-10-22	WooCommerce plugin 1.1.4: automatic reconciliation of pending orders and safer handling of cancelled or changed orders.
2026-10-14	Underpayment tolerance per merchant (default 0 %, exact amount); <code>pay.received</code> , <code>pay.shortfall</code> , <code>pay.shortfallPercent</code> , <code>pay.tolerancePercent</code> and <code>underpaidAccepted*</code> fields; <code>POST /v1/invoices/:id/accept-underpayment</code> .
2026-10-14	Default 60-minute payment window (<code>expiresInMinutes</code> 5–1440); the crypto amount is fixed until expiry.

Date	Change
2026-10-14	Review events and reasons: <code>invoice.review_requested</code> ; <code>reviewReason</code> <code>FUNDS_ON_CLOSED_INVOICE</code> , <code>STORE_CANCELLED_WITH_RECEIPT</code> , <code>DIFFERENT_PAYMENT_METHOD</code> . Exhausted webhooks can be redelivered from Dashboard → Developers.
2026-10-14	At most three coin/network choices per invoice (<code>400 PAYMENT_METHOD_LIMIT</code>); invoice creation 60 requests per minute per IP; request bodies up to 64 KiB.
2026-10-13	Payment records: per-receipt <code>id</code> , <code>currency</code> , <code>network</code> , <code>requiredConfirmations</code> , <code>matchesSelectedAsset</code> , <code>fee</code> , <code>networkFeeRecovered</code> ; <code>paymentDeadline</code> ; events <code>invoice.payment_detected</code> and <code>invoice.payment_orphaned</code> .
2026-10-13	Multi-network: ETH, BNB, TRX, TON and USDT on four networks; per-currency <code>live</code> flag; buyer network fee in <code>pay.networkFee</code> .
2026-10-13	BSC USDT live; test balances isolated from live balances.
2026-10-13	Test and live invoices are separate (each key sees its own mode; one <code>orderId</code> per mode); fee snapshot per invoice; 60-minute grace period.
2026-10-13	<code>pay.remainingAmount</code> ; the wallet URI and simulation use the remaining amount after a partial payment.
2026-10-13	PDF edition of this guide.
2026-10-13	Partner API v1: <code>callbackUrl</code> , <code>buyerPaysFee</code> , <code>price.baseAmount / price.fee</code> , <code>GET /v1/invoices/by-order/:orderId</code> , <code>POST /v1/invoices/:id/simulate</code> , <code>GET /v1/account</code> , <code>GET /v1/currencies</code> , <code>X-Webhook-Event</code> header, OpenAPI spec.

CoinAvio Partner API — generated 2026-10-25 from the published guide. The online version at <https://api.coinavio.com/v1/partner/docs.md> is always authoritative.