Maxy integration guide

Published by Maxy. 12 November 2026. The full text behind maxy.co/docs, which has the short version. Numbers here are this guide's own.

Base URL of the API: `https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api`. Every route takes your key, apart from `/health` and the status page; `POST /sandbox/keys` gives you a test key with no sign-up.

Contents

1 Start here

1.1 What Maxy is
- 1.1a In five sentences
- 1.1b Who this guide is for
- 1.1c Who holds the funds
- 1.1d Glossary terms
- 1.1e Errors on this page

1.2 The short version
- 1.2a What Maxy does
- 1.2b HTTP or SDK
- 1.2c A sandbox key
- 1.2d One full buy
- 1.2e The order states
- 1.2f Webhooks
- 1.2g Errors on this page
- 1.2h Refunds
- 1.2i Going live
- 1.2j Everything else

1.3 How a swap settles in 60 seconds
- 1.3a Errors on this page

1.4 Which path is mine
- 1.4a What does not exist
- 1.4b Errors on this page

1.5 Glossary
- 1.5a Where next

2 Quickstart, 10 minutes

2.1 Quickstart
- 2.1a Create a test key
- 2.1b First quote with curl
- 2.1c Run the example
- 2.1d First sandbox order
- 2.1e See it
- 2.1f Receive your first webhook
- 2.1g The three most common first-run errors
- 2.1h Then, a real order

2.2 The sandbox in five minutes
- 2.2a What it is
- 2.2b Get a test key
- 2.2c A first order
- 2.2d Pick an outcome
- 2.2e Signed webhooks
- 2.2f Limits
- 2.2g Why nothing can be broadcast
- 2.2h Moving to live
- 2.2i Errors on this page

3 Concepts

3.1 Order lifecycle
- 3.1a States
- 3.1b Reading a status
- 3.1c Cancelling
- 3.1d Polling or webhooks
- 3.1e Errors on this page

3.2 Pricing anatomy
- 3.2a Three objects
- 3.2b Amounts
- 3.2c Errors on this page

3.3 A worked quote
- 3.3a A worked quote
- 3.3b The signing shape
- 3.3c Many at once
- 3.3d Errors on this page

3.4 Order size and limits
- 3.4a The rules
- 3.4b Over your key's cap
- 3.4c When Maxy cannot fill a size
- 3.4d Too small
- 3.4e See your own limits

3.4f Errors on this page
3.5 Firm quotes and expiry
3.5a What a firm quote is
3.5b How long it holds
3.5c The refusal
3.5d The loop
3.5e Errors on this page
3.6 Security model
3.6a What the user signs
3.6b What the settler enforces
3.6c What the verifier proves
3.6d What Maxy can and cannot do
3.6e Selling BTC
3.6f The checks the SDK runs before a wallet prompt
3.6g Errors on this page
3.7 Environments and keys
3.7a Your API key
3.7b Publishable keys
3.7c Environments
3.7d See your key
3.7e Errors on this page
3.8 Rate limits, retries and idempotency
3.8a Rate limits
3.8b Retries and idempotency
3.8c Errors on this page
3.9 Versioning and deprecation
3.9a The HTTP API
3.9b The deprecation window
3.9c The SDK
3.9d The route list
3.9e Errors on this page

4 Integration paths

4.1 Paths at a glance
4.1a Fields at a glance
4.1b Errors on this page
4.2 Aggregator or router adapter
4.2a How it fits
4.2b Errors on this page
4.3 Wallet, exchange or on-ramp embed
4.3a How it fits
4.3b Errors on this page
4.4 DeFi or DEX frontend
4.4a How it fits
4.4b Errors on this page
4.5 Backend-only
4.5a How it fits
4.5b Errors on this page
4.6 The widget
4.6a Add it
4.6b The props
4.6c How it behaves
4.6d Content Security Policy
4.6e Errors on this page
4.7 Hosted checkout
4.7a Sign a link
4.7b Open it
4.7c Errors on this page

5 Guides

5.1 Buy BTC from an EVM chain
5.1a Setup
5.1b List pairs
5.1c Show a price
5.1d Get a quote
5.1e Check the order size
5.1f Allowance
5.1g Sign
5.1h Create the order
5.1i Send the escrow (ETH only)
5.1j Watch the order
5.1k Completed
5.1l Errors on this page
5.2 EVM buy: fields and failures
5.2a Quote fields
5.2b Allowance cases
5.2c Failure branches
5.2d Errors on this page
5.3 Gasless buy
5.3a What changes
5.3b Quote
5.3c Sign
5.3d Create the order
5.3e Watch
5.3f If the solver never opens the escrow
5.3g Errors on this page
5.4 Buy BTC from Solana
5.4a The flow
5.4b The same flow over HTTP
5.4c States
5.4d Selling BTC for SOL
5.4e Differences from EVM buys

5.4f Errors on this page
5.5 Sell BTC to a chain
5.5a The flow
5.5b What to show
5.5c Payment rules
5.5d Limits
5.5e Errors on this page
5.6 Sell BTC over HTTP
5.6a The same flow over HTTP
5.6b Errors on this page
5.7 Track orders and receive webhooks
5.7a Polling or webhooks
5.7b Register an endpoint
5.7c What arrives
5.7d Verify the signature
5.7e Delivery
5.7f Local development with no tunnel
5.7g Errors on this page
5.8 Streams and lookup by hash
5.8a A stream instead of polling
5.8b In a browser: a stream ticket
5.8c Status by transaction hash
5.8d Errors on this page
5.9 Reconcile and export orders
5.9a The order list
5.9b Your own ids and pages
5.9c Month-end, step by step
5.9d Errors on this page
5.10 Refunds and failures
5.10a The rule
5.10b Every failure, in one table
5.10c Getting the coins back
5.10d Support script
5.10e Errors on this page
5.11 When a sell goes wrong
5.11a Selling BTC: what can go wrong
5.11b Errors on this page
5.12 Validate recipient addresses
5.12a The call
5.12b Which addresses receive
5.12c Paste and QR scan
5.12d Errors on this page

6 UX guidelines

6.1 Showing a quote
6.1a Showing a quote
6.1b The size of an order
6.1c Errors on this page
6.2 The signing moment
6.2a What to say
6.2b Errors on this page
6.3 Recipient address entry
6.3a Where the BTC goes
6.3b Errors on this page
6.4 Order states and copy
6.4a The words
6.4b Errors on this page
6.5 Failures and refunds
6.5a What to say
6.5b Errors on this page
6.6 Accessibility
6.6a The rules
6.6b Errors on this page
6.7 Brand assets
6.7a Naming Maxy
6.7b The logo
6.7c Errors on this page

7 Security and compliance

7.1 Keys and secrets
7.1a Where each secret lives
7.1b Errors on this page
7.2 Webhook security
7.2a The rules
7.2b Errors on this page
7.3 Content Security Policy
7.3a The hosts
7.3b Errors on this page
7.4 Verifying contract addresses
7.4a How to check
7.4b Errors on this page
7.5 What Maxy does and does not do
7.5a The statement
7.5b Errors on this page
7.6 Partner obligations
7.6a What stays with you
7.6b Errors on this page
7.7 Screening and decline codes
7.7a Today
7.7b Errors on this page
7.8 Reporting an incident
7.8a What to send

7.8b Errors on this page

8 Testing and going live

8.1 Sandbox

8.1a Switching to it

8.1b What is real and what is not

8.1c Nothing to lose

8.1d Walk an order

8.1e Webhooks

8.1f Errors on this page

8.1g What the sandbox does not do

8.2 Sandbox scenarios

8.2a Timing

8.2b Scenarios

8.2c Move an order on

8.2d Errors on this page

8.3 Test scenarios

8.3a Test in the sandbox

8.3b Test small on mainnet

8.3c Errors on this page

8.4 Load and rate-limit behaviour

8.4a What a 429 looks like

8.4b How the SDK handles it

8.4c How much to test

8.4d Errors on this page

8.5 Go-live checklist

8.5a The checklist

8.5b Errors on this page

8.6 Launch review

8.6a What to send

8.6b Errors on this page

9 Operations

9.1 Status and maintenance

9.1a Is Maxy up?

9.1b Incidents and maintenance

9.1c Errors on this page

9.2 Your account

9.2a Your keys and usage

9.2b Statements

9.2c Status

9.2d Errors on this page

9.3 Support and escalation

9.3a When an order needs help

9.3b Errors on this page

9.4 Raising limits

9.4a Ask your contact

9.4b Errors on this page

9.5 Staying current

9.5a Where changes appear

9.5b Errors on this page

9.6 Troubleshooting

9.6a Start with maxy doctor

9.6b Common symptoms

9.6c Errors on this page

10 Reference

10.1 HTTP API

10.1a A price in one line

10.1b Price or quote

10.1c Three ways to name a swap

10.1d Endpoints

10.1e Buying BTC, end to end

10.1f USDC to BTC with no gas

10.1g Selling BTC, end to end

10.1h SOL or an SPL token to BTC, end to end

10.1i Refunds

10.1j BTC to SOL or an SPL token, end to end

10.1k EVM to EVM, end to end

10.1l Quotes last 60 seconds

10.1m Buying an exact amount of BTC

10.1n Status

10.1o Webhooks

10.1p Your account and the status page

10.1q Errors

10.1r Limits

10.1s Versions and changes

10.1t Where next

10.2 API reference

10.2a GET /

10.2b GET /openapi.json

10.2c GET /chains

10.2d GET /tokens

10.2e GET /pairs

10.2f GET /addresses/validate

10.2g GET /limits

10.2h GET /health

10.2i GET /key

10.2j GET /price

10.2k GET /quote

10.2l POST /orders

10.2m POST /orders/submit

10.2n GET /status

10.2o GET /orders/{id}
10.2p GET /orders/{id}/refund
10.2q GET /orders/{id}/deposit
10.2r GET /orders/{id}/claim
10.2s GET /orders/{id}/bridge
10.2t GET /orders/{id}/steps
10.2u POST /orders/{id}/steps
10.2v GET /orders
10.2w POST /orders/watch
10.2x POST /webhooks
10.2y GET /webhooks
10.2z DELETE /webhooks/{id}
10.2aa POST /webhooks/{id}/test
10.2ab GET /webhooks/{id}/events
10.2ac GET /sandbox/scenarios
10.2ad GET /sandbox/rates
10.2ae POST /quotes/batch
10.2af GET /prices/stream
10.2ag POST /prices/stream-ticket
10.2ah POST /orders/{id}/signature
10.2ai POST /orders/{id}/cancel
10.2aj GET /orders/{id}/events
10.2ak GET /orders/{id}/events/stream
10.2al POST /orders/{id}/stream-ticket
10.2am POST /webhooks/{id}/events/{eventId}/replay
10.2an POST /sandbox/keys
10.2ao GET /account
10.2ap GET /usage
10.2aq POST /keys
10.2ar POST /keys/rotate
10.2as POST /keys/{prefix}/origins
10.2at DELETE /keys/{prefix}
10.2au POST /limits/requests
10.2av GET /limits/requests
10.2aw GET /statements
10.2ax GET /checkout/verify
10.2ay GET /explorer/{query}
10.2az POST /feedback
10.2ba POST /security-reports
10.2bb GET /system/status
10.2bc GET /system/history
10.2bd GET /system/incidents
10.2be GET /system/maintenance
10.2bf GET /system/verified
10.2bg GET /system/feed.xml
10.2bh POST /system/subscribe
10.2bi GET /system/unsubscribe
10.2bj GET /system/confirm
10.2bk POST /sandbox/orders/{id}/advance

10.3 SDK reference
10.3a Packages
10.3b Calls and their endpoints
10.3c Failure shape
10.3d Python and curl
10.3e Errors on this page

10.4 The SDK client
10.4a What to import
10.4b The client
10.4c Errors
10.4d Errors on this page

10.5 SDK flow helpers, React and Python
10.5a Flow helpers
10.5b React
10.5c Python
10.5d Testing your integration
10.5e Errors on this page

10.6 Error codes
10.6a HTTP API error codes
10.6b Codes
10.6c Server codes you may see in serverCode
10.6d Where next

10.7 Webhook events
10.7a Events
10.7b Envelope
10.7c order.updated data
10.7d Headers
10.7e Ordering and retries
10.7f Example deliveries
10.7g A real delivery log
10.7h Errors on this page

10.8 Order states
10.8a Each state, from the sandbox
10.8b rawStatus under each state
10.8c Errors on this page

10.9 Contracts
10.9a Settlers
10.9b What is deployed
10.9c Source and versions
10.9d Who can change what
10.9e What the settler guarantees
10.9f Errors on this page

10.10 Chains and tokens

10.10a Read them
10.10b Tokens
10.10c Token list files
10.10d Naming a coin
10.10e Confirmations
10.10f Errors on this page
10.11 Limits
10.11a Errors on this page
10.12 Runnable samples
10.12a Run one
10.12b The files
10.12c The first script
10.12d Errors on this page
10.13 FAQ
10.13a Where next
10.15 Migration guides
10.15a SDK and API 0.15.0: an order ends when the user's funds arrive
10.15b The HTTP API
10.15c The SDK
10.15d Errors on this page

11 Partner program

11.1 How awards work
11.1a The summary
11.1b A worked epoch
11.1c Your records
11.1d Errors on this page

12 Reader tracks

12.1 Source adapter
12.1a Pages, in order
12.1b Done when
12.1c What to send your team
12.1d Errors on this page
12.2 Product embed
12.2a Pages, in order
12.2b Done when
12.2c What to send your team
12.2d Errors on this page
12.3 Frontend button
12.3a Pages, in order
12.3b Done when
12.3c What to send your team
12.3d Errors on this page
12.4 Headless backend
12.4a Pages, in order
12.4b Done when
12.4c What to send your team
12.4d Errors on this page
12.5 Vendor review
12.5a Pages, in order
12.5b Done when
12.5c What to send your team
12.5d Errors on this page

1.1 What Maxy is

After this page you can explain what Maxy does, who holds the money at each moment, and which page to read next.

Before you start: nothing. **Time:** 5 minutes. In a hurry? 1.2 The short version takes you to a completed sandbox order.

1.1a In five sentences

1. Maxy swaps native Bitcoin with coins on Base, Arbitrum, BNB Chain, Avalanche, Robinhood Chain and Solana, through an HTTP API and a TypeScript SDK.
2. The BTC arrives as an ordinary Bitcoin transaction to the user's own Bitcoin address: nothing is wrapped, buying needs no Maxy account, and the user needs no new wallet.
3. Buying BTC takes one signature and one transaction from the wallet the user already has, or only signatures when the user pays in USDC (gasless: a USDC permit and a Permit2 transfer, no gas).
4. Until the BTC is proven paid, the user's coins sit in the settler contract on their own chain, never with Maxy.
5. The solver pays the BTC first, and that contract pays it only against a proof of the payment; if it is late, the user takes their coins back.

1.1b Who this guide is for

You are	You want to	Your track
An aggregator or router engineer	Add Maxy as one more source	12.1 Source adapter
A wallet, exchange or on-ramp engineer	Put a BTC buy inside your product	12.2 Product embed
A DeFi or DEX frontend engineer	Add a Buy BTC button on the wallet connection you have	12.3 Frontend button
A backend or custodial platform engineer	Sign server-side with your own key management	12.4 Headless backend
A security or compliance reviewer	Decide whether Maxy can be a vendor	12.5 Vendor review

1.1c Who holds the funds

Moment	The user's coins	The BTC
Quote, signing	the user's wallet	the solver
Escrow open	the settler contract on the user's chain	the solver
Committed	the settler (plus the solver's bond)	the solver
BTC paid	the settler	the user's Bitcoin address
Proof accepted	the solver, released by the settler	the user
The solver never commits	the user reclaims them after the escrow deadline (2 hours)	never sent
The solver commits and never pays	the user reclaims them, plus the solver's bond, after the Bitcoin deadline (144 blocks, about a day)	never sent

No Maxy server can move the escrow or mark a payment as made: the settler releases funds only against a proof the verifier accepts. Selling BTC runs the other way round: the solver locks the payout in the settler before the user sends any BTC (sell BTC).

1.1d Glossary terms

The full list is in the glossary.

Term	Meaning
Price	What an amount gets right now, for display (`GET /price`). No id, nothing reserved.
Quote	The executable amount for an order about to be placed, with `quoteId` and `expiresAt` (`GET /quote`).
Order	A signed intent sent to Maxy (`POST /orders`). Its id, `orderId`, is the intent hash.
Intent	The EIP-712 message the user signs: who pays, how much, at what price, to which Bitcoin script, until when.
Escrow	The user's coins locked on the settler for one order.
Settler	The contract on each EVM chain that holds escrows and releases them only against a Bitcoin payment proof.
Solver	The party that fills an order: commits with a bond, pays the BTC, proves it. Maxy runs one.
Verifier	The on-chain Bitcoin light client that checks a payment proof and its confirmations.

Term	Meaning
Sats	The smallest unit of BTC: 1 BTC = 100,000,000 sats. Every amount in the API is a string in the smallest unit.
Order size cap	The largest order your API key may place, $100 unless agreed otherwise.

1.1e Errors on this page

None: this page has no call.

Next: 1.2 The short version.

Last verified: 5 November 2026, against the SDK source and the API it runs.

1.2 The short version

After this page you have taken one order from nothing to `completed` in the sandbox, with curl or with the SDK.

Before you start: a test key from Maxy, plus curl, jq and Foundry's cast, or Node 22.18+. **Time:** 15 minutes.

1.2a What Maxy does

- Maxy swaps native Bitcoin with coins on Base, Arbitrum, BNB Chain, Avalanche, Robinhood Chain and Solana.
- The BTC arrives as an ordinary Bitcoin transaction at the user's own address. Nothing is wrapped.
- Until the BTC is proven paid, the user's coins sit in the settler contract on their own chain, never with Maxy.
- The solver pays the BTC first. The contract pays the solver only against a proof of that payment.
- If the solver does not pay in time, the user takes their coins back.

1.2b HTTP or SDK

Use	When	What you get
The HTTP API	Any language, or you want every step in your own code	Plain JSON, and the `steps` to sign and send. Python has the `maxy` client, or calls it with `requests`
The SDK, `@maxy-labs/sdk`	TypeScript	`swap.execute` runs the whole flow and checks each step before the wallet signs. On npm, MIT licence

Both run every route. The sections below show each call both ways.

1.2c A sandbox key

Your test key (`mxk_test_…`) is in your partner kit's README. It runs the same base URL, routes and JSON; nothing real happens and no wallet needs funds. 2.2 The sandbox in five minutes has scenarios and limits.

curl

```
export MAXY_API_KEY=mxk_test_…
export BASE=https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api
export MY_BTC_ADDRESS=bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4           # a test vector:
never send real BTC to it
export TEST_WALLET_KEY=$(cast wallet new --json | jq -r '.[0].private_key')  # a throwaway
wallet
export WALLET=$(cast wallet address --private-key "$TEST_WALLET_KEY")
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/key"
```

TypeScript

```
// mkdir maxy-try && cd maxy-try && npm init -y && npm pkg set type=module
// npm install https://maxy.co/sdk/maxy-labs-sdk-latest.tgz viem
import { generatePrivateKey } from 'viem/accounts';
import { createMaxyClient, evmSignerFromPrivateKey } from '@maxy-labs/sdk';

const maxy = createMaxyClient();                              // reads MAXY_API_KEY; a test key
runs the sandbox
const evm = evmSignerFromPrivateKey(generatePrivateKey());  // a throwaway wallet
const btcAddress = 'bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4';
```

Response


```
{
  "key": "mxk_test_8GOVco…",
  "kind": "secret",
  "scopes": [
    "read",
    "quote",
    "place",
    "orders",
    "webhooks",
    "account"
  ],
  "origins": [],
  "orderCapUsd": 100,
  "sandbox": true
}
```


`GET /key`: a real sandbox answer, 13 November 2026 (`x-request-id` 104dbd34-c9fd-41e7-99ff-daa1f12c9c3b).

1.2d One full buy

This buys BTC with 10 USDC on Base. The route is gasless, so the wallet only signs.

1. Price. For display while the user types. Your key, no wallet, nothing reserved.

curl

```
curl -s "$BASE/price?from=USDC%20on%20base&to=BTC&amount=10"
```

TypeScript

```
const price = await maxy.swap.price({ from: 'USDC on base', to: 'BTC', amount: '10' });
console.log(price.formatted?.receive);
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "10000000",
  "toAmount": "13667",
  "toAmountMin": "13667",
  "formatted": "…",
  "expiresAt": 1794535530,
  "expiresAtIso": "2026-11-13T02:05:30.000Z",
  "limits": {
    "minSats": 546
  },
  "sandbox": true
}
```


`GET /price`: a real sandbox answer, 13 November 2026 (`x-request-id` a4e23a42-1909-467e-acc7-5a24f4d66c99), with long fields folded.

2. Quote. When the user moves to confirm. It names who pays and who receives, and holds for 60 seconds.

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" \
  --data-urlencode "from=USDC on base" -d to=BTC -d amount=10 \
  -d fromAddress="$WALLET" -d toAddress="$MY_BTC_ADDRESS" > quote.json
```

TypeScript

```
// swap.execute, in step 3, gets the quote for you and shows it through onUpdate.
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "10000000",
  "toAmount": "13669",
  "toAmountMin": "13669",
  "formatted": "…",
  "expiresAt": 1794535579,
  "expiresAtIso": "2026-11-13T02:06:19.000Z",
  "limits": "…",
  "quoteId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
  "order": "…",
  "issues": "…",
  "steps": [
    {
      "id": "witness",
      "kind": "signature",
      "description": "Sign the transfer bound to this order. No gas. Sandbox: this signature
authorizes nothing.",
      "signatureKind": "eip712",
      "typedData": "…",
      "hash": "0x1c3aa0811071c34e2f1b536f26ef9ed205ec6eda8598ecaa995d315f8629c707"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signatures. The solver opens the escrow and pays
the gas.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    }
  ],
  "signing": "…",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 686e7eae-9a18-4bba-a051-21694b9fd7d4), with long fields folded.

`quoteId` becomes the order id, and `steps` is what to do next. Live, a USDC quote can also carry a `permit` step: sign every step of kind `signature`, and send a wallet that holds the USDC.

3. Sign and order. The wallet signs, and you post the quote's `order` unchanged with the signature.

curl

```
SIG=$(cast wallet sign --private-key "$TEST_WALLET_KEY" --data "$(jq -c '.steps[] | select(.id
== "witness") | .typedData' quote.json)")
curl -s -X POST "$BASE/orders" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d "$(jq -c --arg sig "$SIG" '{order: .order, signatures: {witness: $sig}}' quote.json)" | tee
placed.json
```

TypeScript

```
const { order, status } = await maxy.swap.execute({
  from: 'USDC on base', to: 'BTC', amount: '10', evm, btcAddress,
  onUpdate: (u) => console.log(u.step, u.step === 'status' ? u.status.state : ''),
});
console.log(order.orderId, status.state, status.btcTxId);
```

Response

```
{
  "orderId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
  "route": "evm-btc-gasless",
  "submitted": true,
  "expiresAt": 1794542720,
  "status": "/status?
orderId=0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea&chainId=8453&route=evm-
btc-gasless",
  "expiresAtIso": "2026-11-13T04:05:20.000Z",
  "refundAt": 1794542720,
  "sandbox": true
}
```

`POST /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` 5c6402da-8cd6-4fc7-b170-d3a79a2b63fe).

4. Status. The order walks `pending`, `in_progress`, `completed` in about 20 seconds in the sandbox.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$(jq -r .orderId placed.json)"
```

TypeScript

```
// swap.execute waits for this; swap.getStatus(order) reads it again later.
```

Response


```
{
  "orderId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
  "route": "evm-btc-gasless",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f58506805d05811929e2096c99c923fa8abbd3e1eb9e1a3083a83",
  "txs": [
    {
      "label": "btc_payout",
      "hash": "53414e44424f58506805d05811929e2096c99c923fa8abbd3e1eb9e1a3083a83"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}` : a real sandbox answer, 13 November 2026 (`x-request-id` 7bf23732-7cb5-43ed-86b4-10228ea3e1ba).

A live `btcTxId` opens at `https://mempool.space/tx/<btcTxId>`. A sandbox one starts `53414e44424f58` and opens nowhere.

1.2e The order states

`state`	Meaning	Who acts next	Final
`pending`	The order is placed; no solver has committed yet	the solver	no
`awaiting_deposit`	Selling: the payout is locked, and the user now sends the BTC	the user	no
`in_progress`	The solver is filling	the solver	no
`completed`	Buying: the BTC was sent. Selling: the payout was released	nobody	yes
`refundable`	Buying: no solver filled in time; the coins wait for the user	the user	no
`refunded`	The coins went back to the user	nobody	yes
`cancelled`	Cancelled before anything moved	nobody	yes

`state`	Meaning	Who acts next	Final
`expired`	Selling and Solana only: expired before it started, or the deposit window closed	nobody	yes
`failed`	Selling and Solana only: `rawStatus` says what went wrong	Maxy support	yes

---

1.2f Webhooks

`POST /webhooks` with `{ "url": "https://your.app/maxy/webhooks" }` registers an endpoint and returns its secret once. Each state change arrives as one signed `order.updated` event. Answer `2xx` within 10 seconds, or Maxy retries 8 times over about 23 hours. Delivery is at least once, so process each event `id` once.

Verify the raw body before you parse it. `Maxy-Signature` is `t=<unix seconds>,v1=<hex>`, where `v1` is HMAC-SHA256 of `<t>.<raw body>`.

TypeScript

```
import { verifyWebhook } from '@maxy-labs/sdk';

// rawBody is the request body as text; verifyWebhook throws on a bad signature or one older
than 5 minutes
const event = await verifyWebhook(rawBody, req.headers['maxy-signature'],
process.env.MAXY_WEBHOOK_SECRET);
if (event.type === 'order.updated') recordOnce(event.id, event.data.orderId, event.data.state);
```

Python

```
import hmac, hashlib, time

def verify(raw_body: bytes, header: str, secret: str, tolerance=300) -> bool:
    parts = dict(p.strip().split("=", 1) for p in header.split(","))
    t, sig = int(parts["t"]), parts["v1"]
    expected = hmac.new(secret.encode(), f"{t}.".encode() + raw_body,
hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, sig) and abs(time.time() - t) <= tolerance
```

1.2g Errors on this page

Every error is `{ "error": { "code", "message", "retryable"?, "docs" }, "requestId" }`. Branch on `code`, and quote `requestId` to Maxy.

Code	HTTP	Cause	Fix	Retry?
`invalid_input`	400	A missing or malformed field	Fix the request; the message names the field	no
`wallet_mismatch`	400	The signature is not the paying wallet's	Sign with the wallet the quote names	no
`auth_required`, `invalid_api_key`	401	No key, or an unknown one	Send a valid `x-maxy-key`	no
`quote_expired`	409	The quote is older than 60 seconds	Get a new quote and sign again	no
`amount_out_of_range`, `order_cap_exceeded`	422	Below 546 sats after costs, or above your key's cap	Change the amount, or ask Maxy to raise the cap	no
`insufficient_liquidity`	422	Maxy cannot fill this order now	A smaller amount, or later	no
`rate_limited`	429	Over 600 calls a minute (120 with a test key)	Wait `Retry-After` seconds	yes
`internal_error`	500	Something failed on Maxy's side	Retry; if it repeats, send Maxy the `requestId`	yes

---

1.2h Refunds

Buying, the user's coins are in their own wallet or in the settler. If no solver fills in time, the order reads `refundable`. No Maxy server is involved: `GET /orders/{id}/refund?chainId=8453` builds the transaction, and the user's wallet sends it. The SDK's call is `swap.refund`.

A solver has 2 hours from signing to commit, then 144 Bitcoin blocks (about a day) to pay. A gasless order with no escrow has nothing to refund: the USDC never left the wallet.

Try it now: add `-d scenario=refund_available` to the quote in step 2.

1.2i Going live

1. Change the key to your `mxk_live_…`. Nothing else changes: same base URL, routes and SDK.
2. Run one small mainnet order: buy BTC with $5 of USDC on Base. Only a real order proves your wallet and chain settings.
3. Check the short list:

- [ ] The live key is in your server's secret store, never in a browser bundle or a repository.

- [ ] Every BTC address is checked with `GET /addresses/validate` before quoting.
- [ ] Your webhook endpoint verifies the signature and processes each event `id` once.
- [ ] Your ops alerting fires on `refundable`, and your users see the refund button.
- [ ] No test key is in production, and your code can never send a sandbox `transaction`.

1.2j Everything else

You need	Read
Your kind of integration	1.4 Which path is mine
Why it is safe	3.6 Security model
Other routes	EVM, Solana, selling BTC
Every route, error and scenario	API, errors, sandbox
The full go-live list	8.5 Go-live checklist

Next: 1.3 How a swap settles in 60 seconds.

Last verified: 13 November 2026, against the sandbox.

1.3 How a swap settles in 60 seconds

After this 60-second read you can draw a swap: the user signs, the escrow holds, the solver pays on Bitcoin, the verifier checks the proof, the escrow releases.

Before you start: 1.1 What Maxy is. **Time:** 1 minute.

Buying BTC with ETH on Base, the reference flow every other flow reuses:

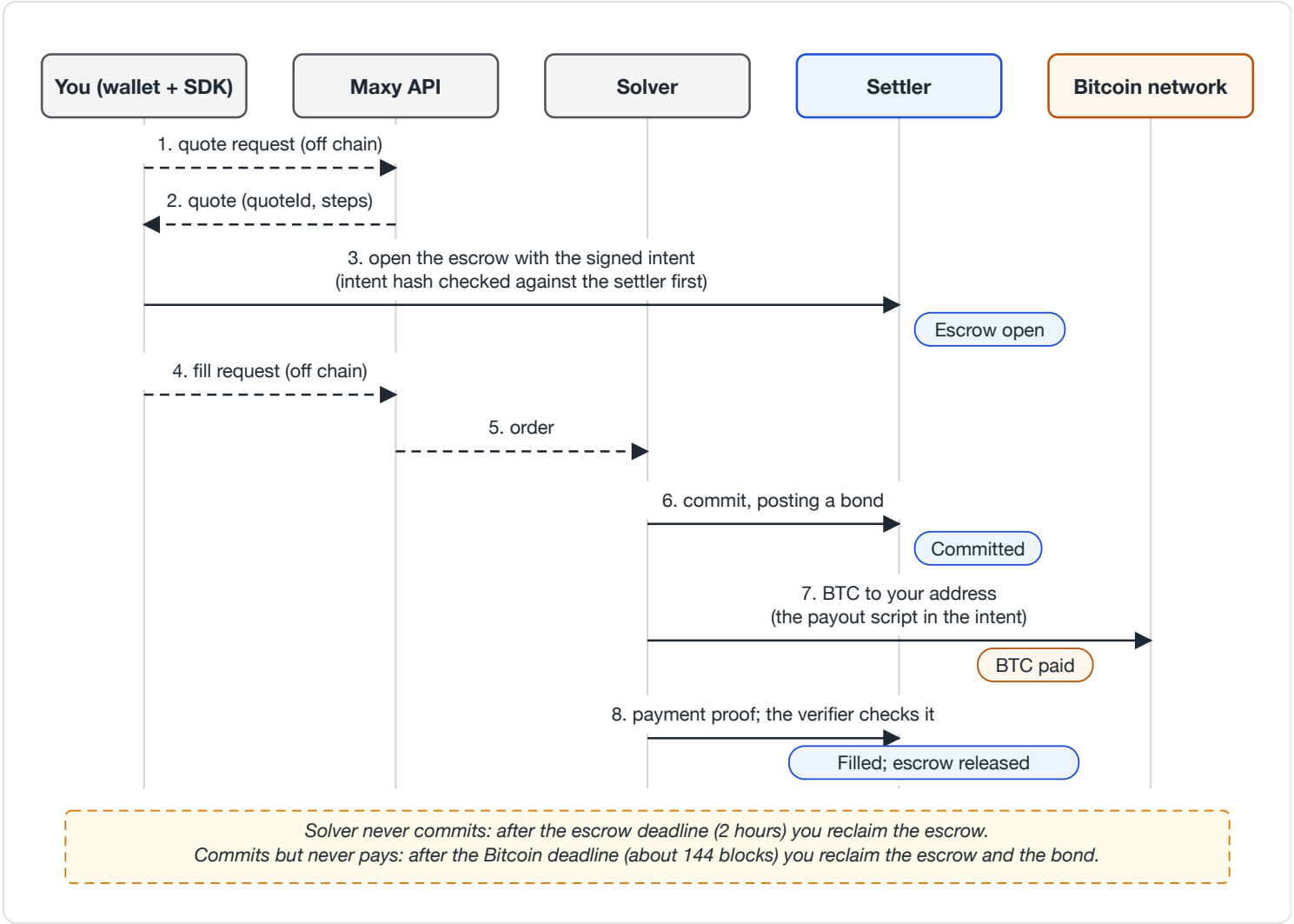


Figure 1. The user's wallet opens an escrow on the settler with the signed intent. The solver commits with a bond, pays the BTC to the user's address, and proves that payment to the verifier, an on-chain Bitcoin light client. Only then does the settler release the escrow to the solver.

Step	What happens	Who acts	Typical time
1. Quote	`GET /quote` returns the amount and the steps to sign	your app	under a second
2. Sign	The user signs the intent: amount, price, recipient script, deadlines	the user's wallet	the user's time

Step	What happens	Who acts	Typical time
3. Escrow	The user's wallet sends one transaction that locks the coins on the settler	the user's wallet	one block
4. Commit	The solver commits to the order on the settler and posts a bond (1% by default)	solver	seconds
5. Pay	The solver sends the BTC to the user's Bitcoin address	solver	Maxy's solver typically within 3 seconds
6. Prove	The solver proves the Bitcoin payment to the verifier after 1 confirmation	solver	about 10 minutes (one Bitcoin block)
7. Release	The settler releases the escrow and the bond to the solver	settler contract	same transaction as 6

The user has the BTC at step 5, and an EVM buy reads `completed` from then on. Steps 6 and 7 are how the solver gets paid; the user waits for neither. At step 4 the settler also returns to the user anything escrowed above the settlement amount (the price band).

1.3a Errors on this page

None: this page has no call. Refusals are on error codes.

Next: 1.4 Which path is mine.

Last verified: 5 November 2026, against the SDK source and the API it runs.

1.4 Which path is mine

After this page you know which track to follow, and what "done" looks like for your kind of integration.

Before you start: 1.1 What Maxy is. **Time:** 2 minutes.

A track is a reading order with a finish line. Pick the row that sounds like you. Every build track ends at the go-live checklist; the vendor review ends at a filled questionnaire.

You are	Your track	Your path	Done when
An aggregator or router adding Maxy as one more source	12.1 Source adapter	4.2 Aggregator or router adapter	Your adapter maps a quote, reads `/pairs`, skips a size it cannot place, and tracks by `txHash`
An exchange, wallet or on-ramp putting a BTC buy in your product	12.2 Product embed	4.3 Wallet, exchange or on-ramp embed	A user buys BTC from the wallet they have, and webhooks update your ledger once per event
A DeFi or DEX frontend adding "Buy BTC"	12.3 Frontend button	4.4 DeFi or DEX frontend	The button ships on your wallet connection, with your key on your server
A custodial platform, neobank or treasury tool signing server-side	12.4 Headless backend	4.5 Backend-only	Orders are signed by your key system, and month-end reconciles without asking Maxy
Security, compliance or procurement deciding on Maxy as a vendor	12.5 Vendor review	7.5 What Maxy does and does not do	Every technical question is answered by a link, or by the list below

---

1.4a What does not exist

Partners look for these. None exists today; each line says what to use instead.

You may look for	Today
A testnet	The sandbox takes its place: a test key, the same API, nothing real. Then a few small mainnet orders (testing)
Keys, usage and statements	The API: `GET /account`, `POST /keys/rotate`, `GET /statements` (your account)
A status page	maxy.co/status
A hosted checkout or a drop-in widget	`@maxy/widget` and hosted checkout
A hosted order explorer for partners	`GET /status?txHash=` finds any order; `btcTxId` opens on `mempool.space`

You may look for	Today
npm and PyPI packages	The TypeScript and Python packages install from https://maxy.co/sdk/; React is in your partner kit (the SDK client)
SOC 2 report, penetration test letter, signed SLA	Not published. The contracts' audit status is on audits

---

1.4b Errors on this page

None: this page has no call.

Next: 1.5 Glossary, then the 2.1 Quickstart.

Last verified: 5 November 2026, against the SDK source and the API it runs.

1.5 Glossary

This page is for every reader: it fixes one name for each concept, and the SDK pages use only these names.

Code identifiers keep the spelling the code uses (for example `ProofMint`, `BitcoinIntentSettler`, `lenderStake`). Prose uses the names below.

Term	Meaning	Not called
Maxy	The company that operates the Maxy API, one solver among others, and the keeper, and publishes this SDK.	
Maxy SDK	This TypeScript library (`@maxy-labs/sdk`). It builds and checks every request and transaction locally and never holds a key.	
Maxy API	The HTTPS API the SDK calls for quotes, swap orders, loan requests, sign-in and order records. Calls carry the API key issued during onboarding (getting started). The literal endpoints are listed for security reviewers in endpoints.	backend
Solver	A party that quotes prices and fills orders: it commits to an order in the settler with a bond, pays out Bitcoin or EVM assets, and is paid by the escrow only against a proof of delivery. Solving is open and permissionless: any solver can join the race, and the first qualified bonded commit fills the order. Maxy runs a solver, which is also the lender in every MaxyLock loan Maxy offers; it is reached through the Maxy API (Solana routes excepted, see buy BTC from Solana).	market maker, MM, bot, liquidity bot
Keeper	Maxy's automated service for MaxyLock, one per host chain. It commits signed loans, submits funding proofs, reveals secrets after repayment, broadcasts the borrower's signed returns and sweeps defaulted Bitcoin. Its health is published as a heartbeat (operations).	
Header relayer	A service that submits Bitcoin block headers to the verifier. Maxy operates one; anyone may run one, and the verifier checks proof of work itself.	
Verifier	The on-chain Bitcoin light client (contract `BitcoinVerifier`). It accepts a proof that a Bitcoin transaction is in a block with enough confirmations. The settler, the MaxyLock loan contract and the Maxy Mint contracts release funds only against such a proof.	SPV contract
Settler	The EVM contract that holds the escrow of an EVM to BTC swap and releases it only against a verifier proof of the Bitcoin payment (contract `BitcoinIntentSettler`). One per host chain.	settlement venue
Intent	A signed EIP-712 order for an EVM to BTC swap: who pays, which asset, how many sats, at what price, to which Bitcoin script, with which deadlines. Its hash is the order id.	
Escrow	The funds a user locks on the settler for an EVM to BTC swap.	

Term	Meaning	Not called
Price	What an amount gets right now, for display while the user types (`GET /price`, `swap.price`). No id, nothing reserved; it takes your key, as every route does.	indicative quote
Quote	The executable amount for an order about to be placed, with `quoteId` and `expiresAt` (`GET /quote`, `swap.quote`). Valid 60 seconds.	firm quote (same thing)
Order	A signed intent sent to Maxy (`POST /orders`, `swap.createOrder`). Its id, `orderId`, is the intent hash; the quote's `quoteId` is the same value.	trade, transaction
Order size cap	The largest order an API key may place, $100 unless agreed otherwise. Prices and quotes answer at any size; the order is refused above it: "Order size is capped for this API key."	limit (alone)
Confirmation	A Bitcoin block on top of (and including) the block holding a payment. The settler needs `requiredConfirmations` (default 1) before it accepts a proof.	
Proof	The raw Bitcoin transaction, its Merkle branch and the block headers above it, which the verifier checks before the settler releases an escrow.	SPV proof
Idempotency key	What makes a retry safe. The order id, which is the intent hash, plays that part; an `Idempotency-Key` header is accepted and has no further effect (retries).	
Secret key	Your integration's key for a server, `mxk_live_…` or `mxk_test_…`. It can do everything the API offers.	
Publishable key	A key for a page, `mxk_pk_live_…` or `mxk_pk_test_…`: prices and quotes only, from the origins it was issued for (keys).	
`rawStatus`	The solver's own status under an order's `state`, for support (order lifecycle).	substatus
Refundable	The order state in which the user can take the escrow back from the settler because a deadline passed.	
Request id	The `x-request-id` header (and `requestId` in error bodies) on every HTTP API answer. Quote it when asking Maxy about a call.	trace id
MaxyLock	Maxy's Bitcoin-backed lending protocol: Bitcoin locked in hash time-locked outputs, a loan paid out on a host chain. Protocol: MaxyLock Documentation.	
Lender's collateral	The loan-asset amount the lender locks in the MaxyLock loan contract for each loan. It pays the borrower if the lender fails to reveal a secret in time. The SDK field is `lenderStake`.	lender stake
Tranche	One Bitcoin output of a MaxyLock loan. Collateral tranches (1 to 7) are released or defaulted one by one; the interest tranche holds the prepaid interest.	
Secret, hash lock	Each collateral tranche is locked to the SHA-256 hash of a 32-byte secret the lender holds. Revealing the secret on the loan contract is what lets the borrower take the Bitcoin back.	preimage (except in code)
Lender key version	The extended public key from which Maxy derives its Bitcoin key for each loan. Compiled into the SDK; a server cannot change it.	epoch (for MaxyLock)

Term	Meaning	Not called
Maxy Mint	Maxy's issuance system for a Bitcoin-backed token on Robinhood Chain. The API and the contracts carry the name **ProofMint**. Protocol: Maxy Mint Documentation.	ProofMint (in prose)
Issuer	The organization that operates Maxy Mint's order signing, minting, refunds and redemption payouts (Maxy Mint 20.2).	
Mint epoch	One configuration period of Maxy Mint: chain, contracts, custody policy, deposit limits and confirmation range.	
BTC.R	The token Maxy Mint issues on Robinhood Chain. Its on-chain ERC-20 symbol is `MAXYBTCb` (18 decimals; 1 sat = 10,000,000,000 units).	MAXYBTCb (in prose)
Direct-custody mode	The live Maxy Mint mode: the Bitcoin goes to a deposit address the issuer controls, and the issuer mints or refunds (Maxy Mint 19.2).	
HTLC mode	The planned Maxy Mint mode in which the deposit sits in a hash time-locked output until the issuer claims it. See mint.	
Base	Base mainnet, chain id 8453. MaxyLock market `'base-mainnet'`; loans in USDC (`0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913`, 6 decimals).	mainnet (as a market id)
Robinhood Chain	Robinhood Chain mainnet, chain id 4663. MaxyLock market `'robinhood-mainnet'`; loans in USDG (`0x5fc5360D0400a0Fd4f2af552ADD042D716F1d168`, 6 decimals). Maxy Mint runs here only.	Robinhood (alone)
Host chain	An EVM chain where Maxy's contracts are deployed: Base, Robinhood Chain, BNB Chain (56), Arbitrum (42161) and Avalanche (43114) for swaps; Base and Robinhood Chain for MaxyLock; Robinhood Chain for Maxy Mint.	
Session	A wallet's signed-in session with the Maxy API, from Sign-In with Ethereum (`auth.signIn`). Required by calls that act for a wallet.	
Smallest unit	Every amount in the SDK: sats for BTC, wei for ETH, 6-decimal units for USDC and USDG, lamports for SOL.	
bps	Basis points: 100 bps = 1%.	
Filler	The MaxyLock chapters' word for a solver: the party that fills an order on the settler.	

The same thing under other names. Pages outside these docs sometimes use other words; they mean the terms above:

You may read	It is
node, light client, SPV, Bitcoin node contract	the **Verifier** (an on-chain Bitcoin light client that checks headers and Merkle proofs)
market maker, filler	a **Solver**
firm quote	a **Quote**

You may read	It is
indicative price	a **Price**
substatus	`rawStatus`
deposit or stake (of the lender)	the **Lender's collateral** (`lenderStake` in the SDK)
vault (of an order)	the **Escrow**: a record in the one **Settler** contract on each chain

---

1.5a Where next

- 25.4 Concepts: how the parts fit, with the architecture figure.
- 25.2 Getting started: install, API key, first calls.
- Documentation index.

Next: 2.1 Quickstart.

Last verified: 7 November 2026, against the SDK source and the API it runs.

2.1 Quickstart

After this page you have a test key working, a first quote, a sandbox order at `completed`, and a signed webhook at your endpoint.

Before you start: your test key and partner kit from Maxy, `curl`, `jq`, and Node 22.18+ (older Node: `npx tsx`) or Python 3.10+. No wallet with funds is needed. **Time:** 10 minutes.

Nothing on this page is real. With a test key (`mxk_test_…`) no money exists and no transaction can be sent (sandbox). Every sample reads your key from `MAXY_API_KEY` and calls the API at `BASE`.

2.1a Create a test key

Get a test key with one call, no sign-up:

`curl`

```
curl -s -X POST "$BASE/sandbox/keys" -H 'content-type: application/json' -d '{"name":"my project"}'
```

It answers `secretKey` and `publishableKey`, once. Your partner kit's README also holds one. Set it and check it:

`curl`

```
export MAXY_API_KEY=mxk_test_…
export BASE=https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/key"
```

Response


```
{
  "key": "mxk_test_8GOVco…",
  "kind": "secret",
  "scopes": [
    "read",
    "quote",
    "place",
    "orders",
    "webhooks",
    "account"
  ],
  "origins": [],
  "orderCapUsd": 100,
  "sandbox": true
}
```


`GET /key`: a real sandbox answer, 13 November 2026 (`x-request-id` 104dbd34-c9fd-41e7-99ff-daa1f12c9c3b).

`sandbox` is `true`, and `orderCapUsd` is the largest order this key may place.

2.1b First quote with curl

A price, with your key (no wallet, nothing reserved):

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/price?from=USDC%20on%20base&to=BTC&amount=10"
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": {
    "chain": 8453,
    "chainName": "Base",
    "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
    "symbol": "USDC",
    "decimals": 6,
    "caip2": "eip155:8453",
    "assetId": "eip155:8453/erc20:0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913"
  },
  "to": {
    "chain": "bitcoin",
    "chainName": "Bitcoin",
    "token": "BTC",
    "symbol": "BTC",
    "decimals": 8,
    "caip2": "bip122:000000000019d6689c085ae165831e93",
    "assetId": "bip122:000000000019d6689c085ae165831e93/slip44:0"
  },
  "fromAmount": "10000000",
  "toAmount": "13667",
  "toAmountMin": "13667",
  "formatted": {
    "send": "10.00 USDC on Base",
    "receive": "0.00013667 BTC",
    "pair": "USDC on Base → BTC"
  },
  "expiresAt": 1794535530,
  "expiresAtIso": "2026-11-13T02:05:30.000Z",
  "limits": {
    "minSats": 546
  },
  "sandbox": true
}
```


`GET /price`: a real sandbox answer, 13 November 2026 (`x-request-id` a4e23a42-1909-467e-acc7-5a24f4d66c99).

`toAmount` is what arrives, in sats, and `formatted.receive` says it in words. A quote names who pays and who receives, and needs your key. Any EVM address works in the sandbox:

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" \
  --data-urlencode "from=ETH on base" -d to=BTC -d amount=0.004 \
  -d fromAddress=0x7d10C4731123a8E14B6f1Cd45860A5c8495cedA0 -d
toAddress=bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
```

Response


```
{
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "4000000000000000",
  "toAmount": "10887",
  "toAmountMin": "10887",
  "formatted": {
    "send": "0.004 ETH on Base",
    "receive": "0.00010887 BTC",
    "pair": "ETH on Base → BTC"
  },
  "expiresAt": 1794535550,
  "expiresAtIso": "2026-11-13T02:05:50.000Z",
  "limits": {
    "minSats": 546
  },
  "quoteId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "order": {
    "route": "evm-btc-escrow",
    "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
    "chainId": 8453,
    "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
    "intent": "…",
    "status": "/status?
orderId=0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3&chainId=8453&route=evm-
btc-escrow",
    "sandbox": "…",
    "quote": {
      "expiresAt": 1794535550,
      "id": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
      "token": "e2860bd87c250703d6079b481f847a2ae01f097e7fca558b97b51a7aced769a3",
      "ask":
"from=ETH+on+base&to=BTC&amount=0.004&fromAddress=0xdd0d268Abc55f328F88f1Ee2e2f0850bFf43686b&toAddr
    }
  },
  "issues": {
    "balance": null,
    "allowance": null,
    "orderCap": null
  },
  "steps": [
    {
      "id": "intent",
      "kind": "signature",
      "description": "Sign the order (EIP-712). No gas. Sandbox: this signature authorizes
```

```
nothing.",
      "signatureKind": "eip712",
      "typedData": {
        "domain": {
          "name": "Maxy Sandbox",
          "version": "1",
          "chainId": 8453,
          "verifyingContract": "0x0000000000000000000000000000000000000000"
        },
        "types": "…",
        "primaryType": "MaxySandboxOrder",
        "message": {
          "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
          "step": "intent",
          "notice": "Maxy sandbox. This moves nothing and cannot be spent."
        }
      },
      "hash": "0x25b9b9f55bfdf01ce7a847c7342f934e59e5c02d58ab9d8af87a17f1cba4773c"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signature; you get back the transaction to send.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    },
    {
      "id": "escrow",
      "kind": "transaction",
      "description": "Sandbox: do not send the transaction. POST /orders/submit with its
simulatedHash.",
      "chainId": 8453,
      "transaction": null
    }
  ],
  "signing": "…",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 6503b124-ab52-4f9b-a6dd-3bcac3d29a36), with long fields folded. Long fields are folded here; [buy BTC from an EVM chain] shows a quote whole.

`quoteId` becomes the order id, `expiresAt` is 60 seconds away, and `steps` lists what the wallet does next.

2.1c Run the example

The SDK installs from maxy.co as `@maxy-labs/sdk`. The kit's `examples/quickstart.ts` and `examples/quickstart.py` print a price, a quote and its steps. Nothing is signed or sent:

curl

```
mkdir maxy-quickstart && cd maxy-quickstart && npm init -y && npm pkg set type=module
npm install https://maxy.co/sdk/maxy-labs-sdk-latest.tgz viem
cp /path/to/kit/examples/quickstart.ts /path/to/kit/examples/quickstart.py .
node quickstart.ts                                          # Node 22.18+; older Node: npx tsx
quickstart.ts
python3 -m venv .venv && .venv/bin/pip install requests eth-account
.venv/bin/python quickstart.py
```

```
$ node quickstart.ts
price   10.00 USDC on Base → 0.00011999 BTC   (evm-btc-gasless)
quote   10.00 USDC on Base → 0.00011999 BTC   from 0x6F8f8a559C2ea48bb837e960419534f446d82DFc to
bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
        quoteId 0x53414e44424f5812e5dba3586ab7ef6d4733bbc40474963afd6cf4e27382de5b   expires
2026-11-04T18:48:38.000Z   request 781ccb89-d0d9-4f73-842b-a73f160126af
step 1  witness  signature    Sign the transfer bound to this order. No gas. Sandbox: this
signature authorizes nothing.
                              EIP-712 digest
0xd6d0cbc31700fb46c80732956ca49496962adf392ee2e9b02af348e6a0fa57d4
step 2  order    post         Send the order and its signatures. The solver opens the escrow and
pays the gas.
        nothing signed or sent: 04-sign.ts and 05-order.ts take it from here

$ FROM='ETH on base' AMOUNT=0.004 node quickstart.ts
price   0.004 ETH on Base → 0.000116 BTC   (evm-btc-escrow)
quote   0.004 ETH on Base → 0.000116 BTC   from 0xe7e17aE4191D7544d4eeC9fb36458Bcd8EE7e244 to
bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
        quoteId 0x53414e44424f58ba4c62b43e0f91df4394f7ddcd96be397d6b78e52f2dddc111   expires
2026-11-04T18:48:38.000Z   request 5e19d8e1-9202-4291-a739-897b6662fcba
step 1  intent   signature    Sign the order (EIP-712). No gas. Sandbox: this signature
authorizes nothing.
                              EIP-712 digest
0x2f2bbff600ec27bebca8babfd7c7f10764f8c73c38de1ce6a65c24c984feb04d
step 2  order    post         Send the order and its signature; you get back the transaction to
send.
step 3  escrow   transaction  Sandbox: do not send the transaction. POST /orders/submit with its
simulatedHash.
        nothing signed or sent: 04-sign.ts and 05-order.ts take it from here
```

What it printed with a test key, 4 November 2026.

2.1d First sandbox order

The kit's `sandbox` samples take one order to `completed` with a throwaway wallet. They run the same calls as a real order: quote, sign, `POST /orders`, `POST /orders/submit`, then read the status.

curl

```
#!/usr/bin/env bash
# One whole escrow order in the sandbox with curl, jq and Foundry's cast. Run it with a test
key: nothing real happens,
# and the wallet is a throwaway made here. It never broadcasts: `transaction.simulatedHash`
stands in for the hash.
#   export MAXY_API_KEY=mxk_test_…
set -euo pipefail
BASE=${MAXY_API_BASE:-https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api}
BTC=bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
KEY=$(cast wallet new --json | jq -r '.[0].private_key')
WALLET=$(cast wallet address --private-key "$KEY")

QUOTE=$(curl -sG -H "x-maxy-key: $MAXY_API_KEY" "$BASE/quote" --data-urlencode "from=ETH on
base" -d to=BTC -d amount=0.001 -d fromAddress="$WALLET" -d toAddress="$BTC" ${SCENARIO:+-d
scenario="$SCENARIO"})
echo "$QUOTE" | jq '{sandbox, route, toAmount, steps: [.steps[]?.id]}'
TYPED=$(echo "$QUOTE" | jq -c '.steps[] | select(.id=="intent") | .typedData |
.types.EIP712Domain = [{name:"name",type:"string"},{name:"version",type:"string"},
{name:"chainId",type:"uint256"},{name:"verifyingContract",type:"address"}]')
SIG=$(cast wallet sign --private-key "$KEY" --data "$TYPED")

PLACED=$(curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" -H 'content-type: application/json'
"$BASE/orders" -d "$(echo "$QUOTE" | jq -c --arg s "$SIG" '{order, signatures: {intent: $s}}')")
echo "$PLACED" | jq '{orderId, transaction, simulation}'
ID=$(echo "$PLACED" | jq -r .orderId)
HASH=$(echo "$PLACED" | jq -r .transaction.simulatedHash)

curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" -H 'content-type: application/json'
"$BASE/orders/submit" -d "$(echo "$QUOTE" | jq -c --arg h "$HASH" --arg s "$SIG" '{order,
signatures: {intent: $s}, txHash: $h}')" | jq .
for _ in 1 2 3 4 5 6 7 8; do
  STATUS=$(curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ID")
  echo "$STATUS" | jq -c '{state, btcTxId, sandbox}'
  [ "$(echo "$STATUS" | jq -r .isComplete)" = true ] && break
  sleep 4
done
```

TypeScript

```
// One whole order in the sandbox with the SDK. Run it with a test key: nothing real happens,
and no wallet of yours is used.
//
//   export MAXY_API_KEY=mxk_test_…
//   node sandbox.ts                       a buy, to completed
//   SCENARIO=refund_available node sandbox.ts
import { generatePrivateKey } from 'viem/accounts';
import { createMaxyClient, evmSignerFromPrivateKey, isMaxyError, type SandboxScenario } from
'@maxy-labs/sdk';

const maxy = createMaxyClient(); // reads MAXY_API_KEY
if (!maxy.sandbox) throw new Error('This sample is for a test key (mxk_test_…).');

const evm = evmSignerFromPrivateKey(generatePrivateKey()); // a throwaway wallet that holds
nothing
const scenario = process.env.SCENARIO as SandboxScenario | undefined;

try {
  const result = await maxy.swap.execute({
    from: process.env.FROM ?? 'USDC on base',
    to: 'BTC',
    amount: process.env.AMOUNT ?? '10',
    evm,
    btcAddress: 'bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4',
    scenario,
    onUpdate: (u) => console.log(u.step, u.step === 'status' ? u.status.state : ''),
  });
  console.log({ orderId: result.order.orderId, state: result.status.state, btcTxId:
result.status.btcTxId });
} catch (error) {
  if (!isMaxyError(error)) throw error;
  console.log({ refused: error.code, message: error.message });
}
```

Python

```
# One whole gasless order in the sandbox with requests and eth_account. Run it with a test key:
nothing real happens, and
# the wallet is a throwaway made here.
#   export MAXY_API_KEY=mxk_test_…
import os
import time

import requests
from eth_account import Account
from eth_account.messages import encode_typed_data

BASE = os.environ.get("MAXY_API_BASE",
"https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api")
HEADERS = {"x-maxy-key": os.environ["MAXY_API_KEY"]}
wallet = Account.create()

quote = requests.get(f"{BASE}/quote", headers=HEADERS, params={
    "from": "USDC on base", "to": "BTC", "amount": "10", "fromAddress": wallet.address,
    "toAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4", "scenario":
os.environ.get("SCENARIO", "success"),
}).json()
assert quote["sandbox"] is True

signatures = {}
for step in quote["steps"]:
    if step["kind"] == "signature":
        typed = step["typedData"]
        signed = Account.sign_message(encode_typed_data(typed["domain"], typed["types"],
typed["message"]), wallet.key)
        signatures[step["id"]] = "0x" + signed.signature.hex().removeprefix("0x")

placed = requests.post(f"{BASE}/orders", headers=HEADERS, json={"order": quote["order"],
"signatures": signatures}).json()
print(placed)
for _ in range(10):
    status = requests.get(f"{BASE}/orders/{placed['orderId']}", headers=HEADERS).json()
    print(status["state"], status["btcTxId"])
    if status["isComplete"]:
        break
    time.sleep(4)
```

What `sandbox.sh` printed:

```
{
  "sandbox": true,
  "route": "evm-btc-escrow",
  "toAmount": "2564",
  "steps": [
    "intent",
    "order",
    "escrow"
  ]
}
{
  "orderId": "0x53414e44424f582e58e592cc7d1844b34798d40c48bae27f1830c40383d86d3a",
  "transaction": {
    "chainId": 8453,
    "from": "0xAf5dF39d870fFE4c2b60b1D87d4694470598822F",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f58a326316277648c476e6f6bde896aef72265a988d5d5e00e02c",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": {
    "ok": true
  }
}
{
  "orderId": "0x53414e44424f582e58e592cc7d1844b34798d40c48bae27f1830c40383d86d3a",
  "route": "evm-btc-escrow",
  "notified": true,
  "status": "/status?
orderId=0x53414e44424f582e58e592cc7d1844b34798d40c48bae27f1830c40383d86d3a&chainId=8453&route=evm-
btc-escrow",
  "sandbox": true
}
{"state":"pending","btcTxId":null,"sandbox":true}
{"state":"pending","btcTxId":null,"sandbox":true}
{"state":"in_progress","btcTxId":null,"sandbox":true}
{"state":"in_progress","btcTxId":null,"sandbox":true}
{"state":"in_progress","btcTxId":null,"sandbox":true}
{"state":"completed","btcTxId":"53414e44424f582c3399933eeb3a5d12ddb1d58c541af81e9a466ae0a901f182","
```


What it printed with a test key, 4 November 2026.

2.1e See it

The sandbox has no block explorer, because nothing is on a chain. Read the order from the API:

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID"
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders"
```

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9",
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13"
    },
    {
      "label": "btc_payout",
      "hash": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 93bd9560-1299-4260-bc3b-351e9c871a10).

`state` moves `pending`, `in_progress`, `completed` over about 20 seconds (order lifecycle). A live order's `btcTxId` opens at `https://mempool.space/tx/<btcTxId>`; a sandbox one starts `53414e44424f58` and opens nowhere.

2.1f Receive your first webhook

Register an HTTPS endpoint you control, then ask Maxy to send it a signed test event:

curl

```
curl -s -X POST "$BASE/webhooks" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d '{ "url": "https://your.app/maxy/webhooks" }'
# store "secret" (shown once), then:
curl -s -X POST "$BASE/webhooks/<id>/test" -H "x-maxy-key: $MAXY_API_KEY"
```

Response

```
{
  "id": "7d320755-9eac-435b-9b4b-d8151856cdde",
  "url": "https://postman-echo.com/post",
  "events": [
    "order.updated"
  ],
  "secret": "whsec_… (shown once: store it)",
  "createdAt": "2026-11-13T02:08:05.553716+00:00",
  "sandbox": true
}
```

`POST /webhooks` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` c93801d0-4315-40ac-8f99-cac312d79550). Registered with a public echo service, then deleted; the secret is masked here.

Response

```
{
  "eventId": "evt_04417bf111cda494f5b148a3",
  "delivered": true,
  "status": 200,
  "error": null,
  "sandbox": true
}
```

`POST /webhooks/{id}/test`: a real sandbox answer, 13 November 2026 (`x-request-id` 7e45d64f-46cc-4eaf-b4a9-d07f7c5d371b).

`delivered` is `true` and `status` is your endpoint's own answer. Your endpoint received a `webhook.test` event signed with that secret; check it as track orders and receive webhooks shows. From now on, every state change of your orders arrives as `order.updated`. On a laptop, `npx maxy webhooks listen --forward-to http://localhost:3000/maxy/webhooks` receives the same signed events with no tunnel (local development).

If something fails, `npx maxy doctor` checks your key, your clock and the connection, and says what to fix (troubleshooting).

2.1g The three most common first-run errors

You see	Cause	Fix
`401 auth_required`	A call without the key (every route but `/health` and the status page)	Add `-H "x-maxy-key: $MAXY_API_KEY"`
`400 invalid_input`: "A quote is for an order about to be placed: pass fromAddress … and toAddress …"	A quote without the two addresses	Add both, or use `/price` to check prices
`issues.orderCap`: "Order size is capped for this API key. Try $100 or less."	The amount is above your key's cap	Order less, or ask Maxy to raise the cap

Response


```
{
  "error": {
    "code": "auth_required",
    "message": "GET /quote takes your Maxy API key in the x-maxy-key header (or Authorization: Bearer mxk_live_…). Every route does, apart from /health, the status page and getting a test key, which needs no sign-up: curl -X POST https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/sandbox/keys",
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#auth_required"
  },
  "requestId": "9ecb2431-4fe8-4b05-a04e-4d180e2d85c3"
}
```


`GET /quote` (HTTP 401): a real sandbox answer, 13 November 2026 (`x-request-id` 9ecb2431-4fe8-4b05-a04e-4d180e2d85c3).

2.1h Then, a real order

Change the key to `mxk_live_…` and hold $5 of USDC on Base in a test wallet. With the SDK, `execute` runs the whole flow. The order is gasless, so the wallet needs no ETH. Over HTTP, follow buy BTC from an EVM chain.

TypeScript

```
import { createMaxyClient, evmSignerFromWalletClient } from '@maxy-labs/sdk';
import { createWalletClient, http } from 'viem';
import { privateKeyToAccount } from 'viem/accounts';
import { base } from 'viem/chains';

const maxy = createMaxyClient(); // reads MAXY_API_KEY
const account = privateKeyToAccount(process.env.TEST_WALLET_KEY as `0x${string}`); // a test
wallet of yours
const evm = evmSignerFromWalletClient(createWalletClient({ account, chain: base, transport:
http() }));

const { order, status } = await maxy.swap.execute({
  from: 'USDC on base', to: 'BTC', amount: '5',
  evm, btcAddress: process.env.MY_BTC_ADDRESS!,
  onUpdate: (u) => console.log(u.step, u.step === 'status' ? u.status.state : ''),
});
console.log(order.orderId, status.btcTxId);
```

Next: 2.2 The sandbox in five minutes.

Last verified: 13 November 2026, against the sandbox.

2.2 The sandbox in five minutes

After this page you can run a whole order with a test key, force the outcomes that are hard to reach, and move to live by changing one value.

Before you start: a terminal. A test key is below. **Time:** 5 minutes.

2.2a What it is

The sandbox is the Maxy API with nothing real behind it. It has the same base URL, the same routes and the same JSON. You change one thing, the key:

Key	Runs	Orders
`mxk_test_…`	The sandbox: every answer has `"sandbox": true` and the header `X-Maxy-Sandbox: true`	Walk their states on a timer. No solver, settler or chain is reached, and no wallet needs funds
`mxk_live_…`	Mainnet	Real swaps, up to your key's order size cap

In the sandbox	
Real	Prices and quote amounts, routes, JSON shapes, error codes, the 60-second quote, your key's order size cap, signed webhooks, signature checks
Not real	Orders, balances and allowances, transactions, deposit addresses, Bitcoin txids

There is no testnet: the sandbox takes its place.

2.2b Get a test key

Get a test key with one call: no sign-up, and it is the sandbox only. The button under any page of these docs gives one too, and fills it into the samples; your partner kit's `README.md` also holds one. It is `mxk_test_` and 32 letters and digits, and Maxy shows it once.

curl

```
export BASE=https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api
export MAXY_API_KEY=$(curl -s -X POST "$BASE/sandbox/keys" -H 'content-type: application/json' -
d '{}' | jq -r .secretKey)
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/key"
```

TypeScript

```
const BASE = 'https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api';
const keys = await fetch(`${BASE}/sandbox/keys`, { method: 'POST', headers: { 'content-type':
'application/json' }, body: '{}' }).then((r) => r.json());
const key = await fetch(`${BASE}/key`, { headers: { 'x-maxy-key': keys.secretKey } }).then((r)
=> r.json());
console.log(keys.secretKey, key.sandbox); // mxk_test_…, true: export it as MAXY_API_KEY
```

Python

```
import requests

BASE = "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api"
secret = requests.post(f"{BASE}/sandbox/keys", json={}).json()["secretKey"]
print(secret, requests.get(f"{BASE}/key", headers={"x-maxy-key": secret}).json()["sandbox"])  #
mxk_test_…, True
```

`GET /key` answers your key's class, its order cap and `"sandbox": true`. With the SDK, `npx maxy doctor` checks the same key, your clock and the connection, and says what to fix.

2.2c A first order

Each of these runs one order to `completed` with a throwaway wallet that holds nothing, in about half a minute, with no SDK and nothing from the kit. The curl one needs `jq` and Foundry's `cast`; the TypeScript one `npm install viem`; the Python one `pip install requests eth-account`. The kit's `examples` folder has the same orders with the SDK (`first-order.ts`, `first-order.py`).

curl

```
#!/usr/bin/env bash
# One whole escrow order in the sandbox with curl, jq and Foundry's cast. Run it with a test
key: nothing real happens,
# and the wallet is a throwaway made here. It never broadcasts: `transaction.simulatedHash`
stands in for the hash.
#   export MAXY_API_KEY=mxk_test_…
set -euo pipefail
BASE=${MAXY_API_BASE:-https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api}
BTC=bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
KEY=$(cast wallet new --json | jq -r '.[0].private_key')
WALLET=$(cast wallet address --private-key "$KEY")

QUOTE=$(curl -sG -H "x-maxy-key: $MAXY_API_KEY" "$BASE/quote" --data-urlencode "from=ETH on
base" -d to=BTC -d amount=0.001 -d fromAddress="$WALLET" -d toAddress="$BTC" ${SCENARIO:+-d
scenario="$SCENARIO"})
echo "$QUOTE" | jq '{sandbox, route, toAmount, steps: [.steps[]?.id]}'
TYPED=$(echo "$QUOTE" | jq -c '.steps[] | select(.id=="intent") | .typedData |
.types.EIP712Domain = [{name:"name",type:"string"},{name:"version",type:"string"},
{name:"chainId",type:"uint256"},{name:"verifyingContract",type:"address"}]')
SIG=$(cast wallet sign --private-key "$KEY" --data "$TYPED")

PLACED=$(curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" -H 'content-type: application/json'
"$BASE/orders" -d "$(echo "$QUOTE" | jq -c --arg s "$SIG" '{order, signatures: {intent: $s}}')")
echo "$PLACED" | jq '{orderId, transaction, simulation}'
ID=$(echo "$PLACED" | jq -r .orderId)
HASH=$(echo "$PLACED" | jq -r .transaction.simulatedHash)

curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" -H 'content-type: application/json'
"$BASE/orders/submit" -d "$(echo "$QUOTE" | jq -c --arg h "$HASH" --arg s "$SIG" '{order,
signatures: {intent: $s}, txHash: $h}')" | jq .
for _ in 1 2 3 4 5 6 7 8; do
  STATUS=$(curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ID")
  echo "$STATUS" | jq -c '{state, btcTxId, sandbox}'
  [ "$(echo "$STATUS" | jq -r .isComplete)" = true ] && break
  sleep 4
done
```

TypeScript

```
// One whole gasless order in the sandbox with fetch and viem, no SDK. Run it with a test key:
nothing real happens, and
// the wallet is a throwaway made here. Node 22.18 or newer runs the .ts file as it is (older:
name it sandbox-fetch.mjs).
//   npm install viem
//   export MAXY_API_KEY=mxk_test_…
import { generatePrivateKey, privateKeyToAccount } from 'viem/accounts';

const BASE = process.env.MAXY_API_BASE ??
'https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api';
const api = async (path: string, body?: unknown) => {
  const res = await fetch(BASE + path, {
    method: body ? 'POST' : 'GET',
    headers: { 'x-maxy-key': process.env.MAXY_API_KEY!, 'content-type': 'application/json' },
    body: body ? JSON.stringify(body) : undefined,
  });
  const json = await res.json();
  if (!res.ok) throw new Error(`${json.error.code}: ${json.error.message}`);
  return json;
};
const wallet = privateKeyToAccount(generatePrivateKey()); // a throwaway wallet that holds
nothing

const quote = await api('/quote?' + new URLSearchParams({
  from: 'USDC on base', to: 'BTC', amount: '10', fromAddress: wallet.address,
  toAddress: 'bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4', scenario: process.env.SCENARIO ??
'success',
}));
if (quote.sandbox !== true) throw new Error('This sample is for a test key (mxk_test_…).');

const signatures: Record<string, string> = {};
for (const step of quote.steps) {
  if (step.kind !== 'signature') continue;
  const { domain, types, primaryType, message } = step.typedData;
  const { EIP712Domain: _domain, ...fields } = types;
  signatures[step.id] = await wallet.signTypedData({ domain, types: fields, primaryType, message
});
}

const placed = await api('/orders', { order: quote.order, signatures });
console.log(placed.orderId, placed.route);
for (let i = 0; i < 10; i++) {
  const status = await api(`/orders/${placed.orderId}`);
  console.log(status.state, status.btcTxId);
  if (status.isComplete) break;
  await new Promise((r) => setTimeout(r, 4000));
}
```

Python

```
# One whole gasless order in the sandbox with requests and eth_account. Run it with a test key:
nothing real happens, and
# the wallet is a throwaway made here.
#   export MAXY_API_KEY=mxk_test_…
import os
import time

import requests
from eth_account import Account
from eth_account.messages import encode_typed_data

BASE = os.environ.get("MAXY_API_BASE",
"https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api")
HEADERS = {"x-maxy-key": os.environ["MAXY_API_KEY"]}
wallet = Account.create()

quote = requests.get(f"{BASE}/quote", headers=HEADERS, params={
    "from": "USDC on base", "to": "BTC", "amount": "10", "fromAddress": wallet.address,
    "toAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4", "scenario":
os.environ.get("SCENARIO", "success"),
}).json()
assert quote["sandbox"] is True

signatures = {}
for step in quote["steps"]:
    if step["kind"] == "signature":
        typed = step["typedData"]
        signed = Account.sign_message(encode_typed_data(typed["domain"], typed["types"],
typed["message"]), wallet.key)
        signatures[step["id"]] = "0x" + signed.signature.hex().removeprefix("0x")

placed = requests.post(f"{BASE}/orders", headers=HEADERS, json={"order": quote["order"],
"signatures": signatures}).json()
print(placed)
for _ in range(10):
    status = requests.get(f"{BASE}/orders/{placed['orderId']}", headers=HEADERS).json()
    print(status["state"], status["btcTxId"])
    if status["isComplete"]:
        break
    time.sleep(4)
```

1. The quote names who pays and who receives. It carries the steps, and `signing` holds what the wallet is asked for.

Response


```
{
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "4000000000000000",
  "toAmount": "10887",
  "toAmountMin": "10887",
  "formatted": "…",
  "expiresAt": 1794535550,
  "expiresAtIso": "2026-11-13T02:05:50.000Z",
  "limits": "…",
  "quoteId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "order": "…",
  "issues": "…",
  "steps": "…",
  "signing": "…",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 6503b124-ab52-4f9b-a6dd-3bcac3d29a36), with long fields folded.

2. The order. The wallet signs the typed data in `signing.payloads`. `POST /orders` then answers the transaction a live order would send. Here it goes to the zero address with `gas` `"1"`. **Do not send it.**

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "expiresAt": 1794542691,
  "transaction": {
    "chainId": 8453,
    "from": "0xdd0d268Abc55f328F88f1Ee2e2f0850bFf43686b",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": "…",
  "next": "…",
  "expiresAtIso": "2026-11-13T04:04:51.000Z",
  "refundAt": 1794542691,
  "sandbox": true
}
```


`POST /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` e6836a36-5e96-48f0-a94d-36486fe8db98), with long fields folded.

3. Submit. `POST /orders/submit` with `transaction.simulatedHash` as `txHash`, where a live order sends the real hash. A gasless order, a sell or an order nobody submits starts by itself, so this step is not always needed.

4. Completed. The order reads `pending`, `in_progress` at 8 seconds and `completed` at 20. `POST /sandbox/orders/{id}/advance` skips the wait.

Response

```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9",
  "txs": "…",
  "failureReason": null,
  "sandbox": true,
  "scenario": "success"
}
```

`GET /status`: a real sandbox answer, 13 November 2026 (`x-request-id` e828f87c-73d8-4eee-8ff3-ea5539d44411), with long fields folded.

2.2d Pick an outcome

Name a scenario on the quote and the order carries it: `scenario` as a query parameter on `GET /price` and `GET /quote`, or in the body of `POST /orders`. A live key ignores it, so your code needs no branch.

curl

```
curl -sG -H "x-maxy-key: $MAXY_API_KEY" "$BASE/quote" --data-urlencode "from=ETH on base" -d
to=BTC -d amount=0.001 \
  -d fromAddress=0x8ba1f109551bD432803012645Ac136ddd64DBA72 -d
toAddress=bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4 -d scenario=over_cap
```

TypeScript

```
const quote = await maxy.quotes.get({ from: 'ETH on base', to: 'BTC', amount: '0.001',
fromAddress, toAddress, scenario: 'over_cap' });
```

Python

```
quote = maxy.quotes.get("ETH on base", "BTC", "0.001", from_address=wallet,
to_address=btc_address, scenario="over_cap")
```

To test	Scenario	What happens
The refund screen	`refund_available`	`in_progress`, then `refundable`; `GET /orders/{id}/refund` answers a refund

To test	Scenario	What happens
A failure	`order_failed`, `order_expired`, `order_cancelled`	The order ends in that state
Underpaying a sell	`underpaid`	`awaiting_deposit`, then `expired`, with the automatic `btc_return` in `txs`
A slow user	`expired_quote`	`POST /orders` answers `409 quote_expired`
Over your cap	`over_cap`	`issues.orderCap` with no steps; the order answers `422`
Your webhook handler	`webhook_replay`	Each event arrives twice with the same `id`

`GET /sandbox/scenarios` lists all of them.

2.2e Signed webhooks

Webhooks are real in the sandbox: the same signature scheme, the same retries, and `"sandbox": true` in each event. Register a public HTTPS endpoint and send it a test event. The answer to the first call has the secret, once.

curl

```
curl -s -X POST "$BASE/webhooks" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" -d '{ "url": "https://your.app/maxy/webhooks" }'
curl -s -X POST "$BASE/webhooks/$WEBHOOK_ID/test" -H "x-maxy-key: $MAXY_API_KEY"
```

TypeScript

```
const hook = await maxy.webhooks.create('https://your.app/maxy/webhooks'); // hook.secret, shown
once
await maxy.webhooks.test(hook.id);
```

Python

```
hook = maxy.webhooks.create("https://your.app/maxy/webhooks")  # hook["secret"], shown once
maxy.webhooks.test(hook["id"])
```

On a laptop, `npx maxy webhooks listen --forward-to http://localhost:3000/maxy/webhooks` gives you the same signed events with no tunnel. Events follow within about 15 seconds of each change, one per state. To check the signature, see track orders and receive webhooks.

2.2f Limits

Limit	Value	Over it
Calls	120 a minute per test key; price reads count apart, at ten times that	`429 rate_limited` with `Retry-After`
Open orders	50 per test key	`429 rate_limited`, until some finish
Order size	Your key's cap, usually $100	`issues.orderCap`, then `422 order_cap_exceeded`

2.2g Why nothing can be broadcast

A test key reaches only the sandbox, and each answer is useless on any chain: a signature authorizes nothing, a transaction goes to the zero address with `gas "1"`, a deposit address is a testnet one. Each carries `simulated: true`, and ids and hashes start with `53414e44424f58`, "SANDBOX" in hex. Why, item by item.

2.2h Moving to live

1. **Change the key** to `mxk_live_…`. The base URL, routes and JSON stay the same.
2. **The one branch stops firing:** `transaction.simulated` is never set on a live order.
3. **Fund a test wallet** with a few dollars, and place a few small mainnet orders. Your live key's order size cap keeps them small.
4. **Go through** the go-live checklist.

TypeScript

```
const hash = transaction.simulated ? transaction.simulatedHash : await
wallet.sendTransaction(transaction);
```

2.2i Errors on this page

Code	HTTP	Cause	Fix
`invalid_api_key`	401	A cut-off key; or a test key sent to a Maxy function other than this API	Check the key; send it to the base URL above
`not_found`	404	A real order id read with a test key: it sees only sandbox orders	Use the sandbox order's id
`quote_expired`	409	More than 60 seconds between the quote and the order, as live	Get a new quote and sign that
`rate_limited`	429	120 calls a minute, or 50 open orders	Wait `Retry-After` seconds, or for orders to finish
`sandbox_refused`	403	With the SDK: a loan, a mint or a sign-in, which have no sandbox form	Swaps only

Next: 3.1 Order lifecycle.

Last verified: 13 November 2026, against the sandbox.

3.1 Order lifecycle

After this page you know every state an order can be in, who acts next, how long it lasts, and which are final.

Before you start: 1.1 What Maxy is. **Time:** 5 minutes.

3.1a States

`GET /status` (and `swap.getStatus`, and every `order.updated` webhook) reports one `state`, the same nine values on every route. `rawStatus` carries the solver's own status underneath, for support.

```
buying BTC   pending ──► in_progress ──► completed        (btcTxId set: the BTC was sent)
                │              │
                └──────────────┴──► refundable ──► refunded

selling BTC  pending ──► awaiting_deposit ──► in_progress ──► completed
                                  └──► expired

Solana       pending ──► in_progress ──► completed, or expired, cancelled, refunded
selling, Solana  failed (rare: read rawStatus). An EVM buy never reads failed or expired: it
reads refundable.
```

`state`	Meaning	Who acts next	Typical time	Final
`pending`	The order is placed: the escrow is landing, or no solver has committed yet	the solver	seconds	no
`awaiting_deposit`	Selling: the payout is locked in the settler for the user, who now sends the BTC	the user	until the deposit window closes (live orders: 19 Bitcoin blocks, about 3 hours)	no
`in_progress`	The solver is filling: committed, paying. Selling: the user's BTC payment was seen	the solver (selling: the proof, after the confirmations)	buying: seconds to a minute; selling: one confirmation, then about a minute	no

`state`	Meaning	Who acts next	Typical time	Final
`completed`	Buying: the BTC was sent to the user's address (`btcTxId` is set). Selling: the payout was released to the user	nobody		yes
`refundable`	Buying: no solver filled before a deadline; the escrow waits in the settler for the user	the user (refund)	until the user sends the refund	no, but nothing happens without the user: `waitForCompletion` stops here
`refunded`	The coins went back to the user	nobody		yes
`cancelled`	Cancelled before anything moved	nobody		yes
`expired`	Selling and Solana only: expired before it started, or the deposit window closed	nobody (selling with a late or mismatched payment: Maxy support)		yes
`failed`	Selling and Solana only: something went wrong; `rawStatus` says what	Maxy support		yes

An order is done for the user when their funds arrive. A buy, EVM or Solana, reads `completed` the moment `btcTxId` is set, and the BTC is on its way to their address (their wallet shows it after the first confirmation, about 10 minutes). A sale reads `completed` when the settler releases the payout to them. Coin to coin reads `completed` when the coins reach the destination chain.

The settlement that follows a buy, the proof and the release of the escrow to the solver, is Maxy getting its own money back. It needs nothing from the user and nothing waits for it: SDK helpers, hooks, webhooks and status polling all end at `completed`. If it runs late or fails, the order still reads `completed`. `rawStatus` keeps moving (`BTC_SENT` then `SETTLED`, or `PAID` then `SETTLED` on Solana) for an operations team that wants to watch it.

An order id Maxy has not seen yet reads `pending` (an escrow is indexed within seconds of landing), never `404`.

3.1b Reading a status

curl

```
curl -s "$BASE/status?orderId=$ORDER_ID&chainId=8453"
```

TypeScript

```
const status = await fetch(`${BASE}/status?orderId=${orderId}&chainId=8453`).then((r) =>
r.json());
```

Python

```
status = requests.get(f"{BASE}/status", params={"orderId": order_id, "chainId": 8453}).json()
```

Response


```
{
  "orderId": "0xed3985cf8b6dcc9a30448c9faaefbaf3a2f2ef9c06f50ce1f46dd955212ce2ea",
  "route": "evm-btc-gasless",
  "state": "completed",
  "rawStatus": "SETTLED",
  "isComplete": true,
  "btcTxId": "6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea3f",
  "txs": [
    {
      "label": "escrow",
      "hash": "0xf103819940fef19e88f9c0eafb24a52bad8de86c277e18e8e2e60a8594c7f8d1"
    },
    {
      "label": "btc_payout",
      "hash": "6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea3f"
    },
    {
      "label": "settlement",
      "hash": "0xef66f624a34cff33a5d5660e1666e15669bc02dfc8b0a27fa827e8b268e702f2"
    }
  ]
}
```


`GET /status` for a real completed gasless order: a real answer from the live API, 2 November 2026 (`x-request-id` eb15599f-74c7-4c77-95e2-8f50240f3b0a).

Field	Meaning
`orderId`	The order id
`state`	One of the nine states above
`rawStatus`	The solver's own status, for support

Field	Meaning
`isComplete`	`true` in a final state (`completed`, `refunded`, `cancelled`, `expired`, `failed`)
`btcTxId`	The Bitcoin payment to the user, once sent (buys)
`txs`	The order's transactions, labelled `escrow`, `commit`, `btc_payout`, `settlement`, `refund`: EVM hashes with `0x`, the Bitcoin txid bare

Before the payment confirms, the solver may re-send it with a higher Bitcoin fee (replace-by-fee): same amount, same address, a new txid. `btcTxId` then changes once; show the latest.

`rawStatus` under each state is on order states.

3.1c Cancelling

While an order is `pending` on an EVM to BTC route, its maker can take the escrow back. `POST /orders/{id}/cancel?chainId=8453` answers the settler call for the wallet to send; Maxy sends nothing. Once a solver has committed there is nothing to cancel: the order pays out, or refunds after its deadline, and the call answers `409 order_state` with `allowedStates`. A gasless order has no escrow of the user's to take back, and lapses at its deadline if the solver does not open one.

curl

```
curl -s -X POST "$BASE/orders/$ORDER_ID/cancel?chainId=8453" -H "x-maxy-key: $MAXY_API_KEY"
```

TypeScript

```
const { transaction } = await maxy.orders.cancel(orderId, { chainId: 8453 });
```

Python

```
transaction = maxy.orders.cancel(order_id, chain_id=8453)["transaction"]
```

Response


```
{
  "orderId": "0x53414e44424f58d83c861d140370c484eb094547ddc05b8e7ac093b585326213",
  "state": "pending",
  "cancellable": true,
  "function": "cancelBuyEscrow",
  "allowedStates": [
    "pending"
  ],
  "explanation": "Takes back the escrow: no solver has committed yet. Sandbox: nothing to send;
the order now reads cancelled.",
  "transaction": "…",
  "sandbox": true
}
```


`POST /orders/{id}/cancel`: a real sandbox answer, 13 November 2026 (`x-request-id` 3b4d6287-ca5d-4059-a2cc-e952d9efaf20), with long fields folded.

Response


```
{
  "error": {
    "code": "order_state",
    "message": "The order is cancelled: nothing to cancel.",
    "details": {
      "state": "cancelled",
      "allowedStates": [
        "pending"
      ]
    },
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#order_state"
  },
  "requestId": "f4e7a63d-8ff4-4c6a-b422-7dc1a8a18aa4",
  "sandbox": true
}
```


`POST /orders/{id}/cancel` (HTTP 409): a real sandbox answer, 13 November 2026 (`x-request-id` f4e7a63d-8ff4-4c6a-b422-7dc1a8a18aa4).

`GET /orders/{id}` also gives `refundAt`, in unix seconds, and `refundAfterBlock` once a solver has committed: when the deposit can be taken back. After a commit the time is estimated from the Bitcoin block the solver must pay by.

3.1d Polling or webhooks

Webhooks for the record, polling while a user watches: track orders and receive webhooks. Poll every 2 seconds while `pending` or `in_progress`, every 30 seconds in `awaiting_deposit`, and stop at a final state or at `refundable`. The SDK's `waitForCompletion` polls every 4 seconds and gives up after 30 minutes (the order carries on).

3.1e Errors on this page

Code	HTTP	Cause	Fix
`invalid_input`	400	`orderId` is not an order id (`0x` and 64 hex digits), or `txHash` is not a 32-byte hash	Send the id your quote or order returned
`not_found`	404	No order has that transaction (`?txHash=`)	Check the hash; a brand-new one may need a few seconds

Next: 3.2 Pricing anatomy.

Last verified: 13 November 2026, against the sandbox.

3.2 Pricing anatomy

After this page you know which call to make while the user types, which when they confirm, and what each amount means.

Before you start: 3.1 Order lifecycle. **Time:** 5 minutes.

3.2a Three objects

	Price	Quote	Order
Call	`GET /price` · `swap.price()`	`GET /quote` · `swap.quote()`	`POST /orders` · `swap.createOrder()`
When	While the user browses or types an amount	When the user is about to confirm	After the user signed
Needs	The pair and an amount	Also `fromAddress` (who pays), `toAddress` (who receives) and your key	The quote's `order` and the signatures
Gives	`toAmount`, `formatted`	The same amounts plus `quoteId`, `expiresAt`, `issues` and the `steps` to sign and send	`orderId`, the escrow transaction to send, and its simulation
Id	none	`quoteId` (equals the order id it becomes)	`orderId`: the intent hash, the same on chain
Reserves anything	no	no	the escrow does, once the transaction lands
Key	no	yes	yes

Call the price as often as you like, and the quote once, when the user is about to confirm.

curl

```
curl -s "$BASE/price?from=ETH%20on%20base&to=BTC&amount=0.004"
```

TypeScript

```
const price = await fetch(`${BASE}/price?from=ETH%20on%20base&to=BTC&amount=0.004`).then((r) => r.json());
```

Python

```
price = requests.get(f"{BASE}/price", params={"from": "ETH on base", "to": "BTC", "amount": "0.004"}).json()
```

Response


```
{
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "4000000000000000",
  "toAmount": "10886",
  "toAmountMin": "10886",
  "formatted": {
    "send": "0.004 ETH on Base",
    "receive": "0.00010886 BTC",
    "pair": "ETH on Base → BTC"
  },
  "expiresAt": 1794535530,
  "expiresAtIso": "2026-11-13T02:05:30.000Z",
  "limits": {
    "minSats": 546
  },
  "sandbox": true
}
```


`GET /price`: a real sandbox answer, 13 November 2026 (`x-request-id` b92d480f-f12b-43af-91bc-34fc4d82b364), with long fields folded.

3.2b Amounts

- Every amount is a decimal string in the smallest unit: sats for BTC, wei for ETH and BNB, 6-decimal units for USDC, lamports for SOL. The `from` and `to` objects carry `decimals` beside them.
- `formatted` has the same amounts in words (`"0.00025282 BTC"`) for display.
- `toAmount` is what arrives at the user's address. It is the executable amount, so `toAmountMin` equals it: the signed intent's price band (`slippageBps`, 0.5% by default) covers price moves between the quote and the escrow.
- Paying with ETH, BNB or AVAX, the escrow transaction's `value` is `fromAmount` plus that band. When the solver commits, the settler returns everything above the settlement amount, so the user pays at most `fromAmount`. The wallet must hold the band (and gas) for a moment.
- Nothing is added or taken after the user signs.

3.2c Errors on this page

Code	HTTP	Cause	Fix
`invalid_input`	400	A quote without `fromAddress` or `toAddress`, or an amount that is not a number	Send both addresses; use `/price` to check prices
`unsupported_route`	400	The pair is not one Maxy swaps	Pick from `GET /pairs`
`amount_out_of_range`	422	Below the smallest order	Raise the amount to at least `limits.minSats`
`order_cap_exceeded`	422	Above your key's order size cap	Order less, or ask Maxy to raise the cap
`quote_expired`	409	The quote's `expiresAt` passed, or the intent's deadline	Get a new quote

The full list, with what each code means for a retry, is in error codes.

Next: 3.3 A worked quote.

Last verified: 13 November 2026, against the sandbox.

3.3 A worked quote

After this page you can reconcile every amount in a real quote, read `signing`, and ask for many quotes at once.

Before you start: 3.2 Pricing anatomy. **Time:** 5 minutes.

3.3a A worked quote

A real quote for 0.004 ETH on Base (`x-request-id` 802219f5-0857-4b74-a9d2-a3ad5b458a24, 2 November 2026), shown in full on buy BTC from an EVM chain:

Line	Field	Value	How it reconciles
The user pays	`fromAmount`	4,000,000,000,000,000 wei	0.004 ETH, as asked
The user receives	`toAmount`, `order.intent.satAmount`	11,259 sats	The executable amount, with every cost in. `toAmountMin` is the same
The signed price	`order.intent.limitPrice`	355,182,427,036	Wei per sat
The settlement amount	`satAmount × limitPrice`	3,998,998,945,998,324 wei	0.00399899 ETH. This is what the solver is paid, and it is at most `fromAmount`
The price band	`order.intent.slippageBps`	50	0.5%
What the wallet sends	`issues.balance.expected`, then `transaction.value`	4,018,993,940,726,289 wei	The settlement amount × 1.005
What comes back	`transaction.value` minus the settlement amount	19,994,994,727,965 wei	When the solver commits, the settler returns everything above the settlement amount

A USDC order works the same way: the Permit2 transfer the wallet signs covers the settlement amount plus the band.

3.3b The signing shape

A firm quote says what the wallet is asked for in one shape on every route: `signing.kind` (`eip712`, `solana_message`, `solana_tx`, `evm_tx`, `bitcoin_payment`), `payloads` to hand the wallet as they are, `approval` if a one-time approval must confirm first, and `after`, the next call. `steps` stays the

full list; `signing` is the part a wallet integration reads. `null` when the quote cannot become an order. On `solana_tx`, `evm_tx` and `bitcoin_payment` the payload comes back from `POST /orders`.

Response


```
{
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "4000000000000000",
  "toAmount": "10884",
  "toAmountMin": "10884",
  "formatted": "…",
  "expiresAt": 1794535802,
  "expiresAtIso": "2026-11-13T02:10:02.000Z",
  "limits": "…",
  "quoteId": "0x53414e44424f5823936eadc65c68429f1fc465d671b4382365c5ff83ffbf13ae",
  "order": "…",
  "issues": "…",
  "steps": "…",
  "signing": {
    "kind": "eip712",
    "payloads": [
      {
        "id": "intent",
        "typedData": {
          "domain": {
            "name": "Maxy Sandbox",
            "version": "1",
            "chainId": 8453,
            "verifyingContract": "0x0000000000000000000000000000000000000000"
          },
          "types": {
            "MaxySandboxOrder": [
              {
                "name": "orderId",
                "type": "string"
              },
              {
                "name": "step",
                "type": "string"
              },
              {
                "name": "notice",
                "type": "string"
              }
            ]
          },
          "primaryType": "MaxySandboxOrder",
          "message": {
            "orderId": "0x53414e44424f5823936eadc65c68429f1fc465d671b4382365c5ff83ffbf13ae",
```

```
            "step": "intent",
            "notice": "Maxy sandbox. This moves nothing and cannot be spent."
          }
        },
        "hash": "0xa5ca31f75242642d0d412ed3de6163391054a597afe9fcaef4b2194cafbd51ab"
      }
    ],
    "approval": null,
    "after": {
      "method": "POST",
      "path": "/orders"
    }
  },
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 598c7f45-0dc2-4d6f-a32b-c5a40776d396), with long fields folded.

3.3c Many at once

`POST /quotes/batch` takes up to 20 requests in one call. A request naming both addresses is a firm quote, which takes a key; the others are prices. One failing does not fail the rest: each result has its own `status`, `requestId`, and `quote` or `error`.

curl

```
curl -s -X POST "$BASE/quotes/batch" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d '{"requests":[{"from":"ETH on base","to":"BTC","amount":"0.004"},{"from":"PEPE on
base","to":"BTC","amount":"1"}]}'
```

TypeScript

```
const { results } = await maxy.quotes.batch([{ from: 'ETH on base', to: 'BTC', amount: '0.004'
}, { from: 'USDC on base', to: 'BTC', amount: '10' }]);
```

Python

```
results = maxy.quotes.batch([{"from": "ETH on base", "to": "BTC", "amount": "0.004"}])
["results"]
```

Response


```
{
  "results": [
    {
      "index": 0,
      "status": 200,
      "requestId": "0cf5d4fd-1d71-461e-ac64-9bbba4d323d8",
      "quote": "…"
    },
    {
      "index": 1,
      "status": 200,
      "requestId": "ac3faae4-3d5c-42f1-9a43-12305c32c7df",
      "quote": "…"
    },
    {
      "index": 2,
      "status": 400,
      "requestId": "73b1d8bf-c950-4f78-ab64-38397ffc7e85",
      "error": {
        "code": "invalid_input",
        "message": "PEPE is not a coin Maxy knows on Base (it knows ETH, USDC, USDT, WBTC,
cbBTC, MORPHO, USDe). Pass its contract address instead.",
        "retryable": false,
        "docs": "https://maxy.co/docs/sdk-errors#invalid_input"
      }
    }
  ]
}
```


`POST /quotes/batch`: a real sandbox answer, 13 November 2026 (`x-request-id` b2cf35b7-32eb-4af2-9a91-95355871ece8), with long fields folded.

3.3d Errors on this page

Code	HTTP	Cause	Fix
`invalid_input`	400	A batch over 20 requests, or a request with one address missing	Fix the request
`auth_required`	401	A firm quote in a batch with no key	Send `x-maxy-key`

Next: 3.4 Order size and limits.

Last verified: 13 November 2026, against the sandbox.

3.4 Order size and limits

After this page you know the smallest order, what your key allows, what the user sees above it, and how to keep the screen useful.

Before you start: 3.2 Pricing anatomy. **Time:** 5 minutes.

3.4a The rules

Rule	Where it shows	What the user sees
Smallest order: 546 sats of BTC, after costs	`limits.minSats` on every BTC buy price and quote; `amount_out_of_range` below it	"Too small: orders start at 546 sats after costs. Try a larger amount."
Your key's order size cap ($100 unless agreed otherwise)	`issues.orderCap` on the quote, with `steps: null`; `order_cap_exceeded` (HTTP 422) from `POST /orders` and `createOrder`	"Order size is capped for this API key. Try $100 or less."

Prices and quotes answer at any size, so the user always sees what an order would get. The cap is checked when the user moves to place the order, before anything is signed. `GET /key` returns your cap.

3.4b Over your key's cap

The quote still shows the amounts, carries `issues.orderCap` and has no steps to sign:

Response


```
{
  "quoteId": null,
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "4000000000000000",
  "toAmount": "10887",
  "toAmountMin": "10887",
  "formatted": {
    "send": "0.004 ETH on Base",
    "receive": "0.00010887 BTC",
    "pair": "ETH on Base → BTC"
  },
  "expiresAt": 1794535546,
  "expiresAtIso": "2026-11-13T02:05:46.000Z",
  "limits": {
    "minSats": 546
  },
  "order": null,
  "issues": {
    "balance": null,
    "allowance": null,
    "orderCap": {
      "capUsd": 100,
      "orderUsd": 150,
      "message": "Order size is capped for this API key. Try $100 or less."
    }
  },
  "steps": null,
  "signing": "…",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 9b4d5cb7-ef94-40c2-b6d3-cb4caf460432), with long fields folded.

Offer the fix on the same screen: "Use $100" fills in `issues.orderCap.capUsd`. Never leave the user on a dead end.

An order sent anyway is refused before anything is signed or sent:

Response


```
{
  "error": {
    "code": "order_cap_exceeded",
    "message": "Order size is capped for this API key. Try $100 or less.",
    "details": {
      "capUsd": 100,
      "orderUsd": 150
    },
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#order_cap_exceeded"
  },
  "requestId": "7bddbf63-2908-4fd5-b157-f371185d4fb2",
  "sandbox": true
}
```


`POST /orders` (HTTP 422): a real sandbox answer, 13 November 2026 (`x-request-id` 7bddbf63-2908-4fd5-b157-f371185d4fb2).

3.4c When Maxy cannot fill a size

Prices answer at any size, with the same rate. A quote for an order Maxy cannot fill now answers `200` with `liquidityAvailable: false` and no steps. No figure is named, so show the message and let the user try a smaller amount.

Response


```
{
  "quoteId": null,
  "route": "evm-btc-escrow",
  "liquidityAvailable": false,
  "from": "…",
  "to": "…",
  "fromAmount": "4000000000000000",
  "toAmount": "10887",
  "toAmountMin": "10887",
  "formatted": {
    "send": "0.004 ETH on Base",
    "receive": "0.00010887 BTC",
    "pair": "ETH on Base → BTC"
  },
  "expiresAt": 1794535547,
  "expiresAtIso": "2026-11-13T02:05:47.000Z",
  "limits": "…",
  "order": null,
  "issues": {
    "balance": null,
    "allowance": null,
    "orderCap": null
  },
  "steps": null,
  "signing": "…",
  "message": "Unfillable right now: this order is larger than Maxy can fill at the moment. Try a smaller amount.",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` afe5d263-3aba-4efa-bb90-6ed973677c7c), with long fields folded.

Response


```
{
  "error": {
    "code": "insufficient_liquidity",
    "message": "Unfillable right now: this order is larger than Maxy can fill at the moment. Try a smaller amount.",
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#insufficient_liquidity"
  },
  "requestId": "3b41ad4e-3fdf-4e52-ad6a-4c272df329da",
  "sandbox": true
}
```

`POST /orders` (HTTP 422): a real sandbox answer, 13 November 2026 (`x-request-id` 3b41ad4e-3fdf-4e52-ad6a-4c272df329da).

An aggregator skips Maxy for that size and ranks the next source (aggregator path).

3.4d Too small

Response

```
{
  "error": {
    "code": "amount_out_of_range",
    "message": "Too small: orders start at 546 sats after costs.",
    "details": {
      "minSats": 546
    },
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#amount_out_of_range"
  },
  "requestId": "385e7fab-9ef7-431a-983d-a5be6068ea41",
  "sandbox": true
}
```

`GET /quote` (HTTP 422): a real sandbox answer, 13 November 2026 (`x-request-id` 385e7fab-9ef7-431a-983d-a5be6068ea41).

3.4e See your own limits

`GET /key` and `GET /limits` answer what this key may do:

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/limits"
```

TypeScript

```
const limits = await fetch(`${BASE}/limits`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
limits = requests.get(f"{BASE}/limits", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "kind": "secret",
  "scopes": [
    "read",
    "quote",
    "place",
    "orders",
    "webhooks",
    "account"
  ],
  "origins": [],
  "orderCapUsd": 100,
  "minOrderSats": 546,
  "routes": {
    "evm-btc-escrow": {
      "minSats": 546
    },
    "evm-btc-gasless": {
      "minSats": 546
    },
    "btc-evm": {
      "minUsd": 10
    },
    "sol-btc": {
      "minUsd": 2
    },
    "btc-sol": {
      "minUsd": 2
    }
  },
  "quoteTtlSeconds": 60,
  "batchMax": 20,
  "rateLimit": {
    "limit": 120,
    "remaining": 118,
    "resetSeconds": 60,
    "per": "key",
    "priceLimit": 1200
  },
  "sandbox": true
}
```


`GET /limits`: a real sandbox answer, 13 November 2026 (`x-request-id` 2a9d335f-ceff-43fc-9c3d-5b741f61a402).

3.4f Errors on this page

Code	HTTP	Cause	Fix
`amount_out_of_range`	422	Below the smallest order	Raise the amount to at least `limits.minSats`
`order_cap_exceeded`	422	Above your key's order size cap	Order less, or ask Maxy to raise the cap
`insufficient_liquidity`	422	Maxy cannot fill this order now	A smaller amount, or later

Next: 3.5 Firm quotes and expiry.

Last verified: 13 November 2026, against the sandbox.

3.5 Firm quotes and expiry

After this page you know how long a quote holds, how to count it down, and what happens when it runs out.

Before you start: 3.2 Pricing anatomy. **Time:** 5 minutes.

3.5a What a firm quote is

A quote (`GET /quote`) is the executable amount for one order: who pays, who receives, how much, with `quoteId` and `expiresAt`. The docs and the API call it a quote; "firm quote" means the same thing.

3.5b How long it holds

- A quote's price holds for 60 seconds: `expiresAt` is Unix seconds. Show a countdown from it and get a new quote when it runs out (UX guidelines).
- An order from an expired quote is refused, before anything is sent: `createOrder` in the SDK, and `POST /orders` over HTTP (`409 quote_expired`). Sign and post within the 60 seconds.
- Once signed, the intent carries its own deadlines: a solver must commit within 2 hours of signing, and pay the BTC within 144 Bitcoin blocks of that commit. Otherwise the user takes the escrow back (refunds). `POST /orders` returns that deadline as `expiresAt` .

A firm quote's `expiresAt` is unix seconds, and `expiresAtIso` is the same moment in RFC 3339. Count down against the `Date` header of Maxy's answers, not the device's clock: `quotes.msUntilExpiry(quote)` does, and `useQuote` gives `secondsLeft` .

3.5c The refusal

Response


```
{
  "error": {
    "code": "quote_expired",
    "message": "This quote expired (a quote lasts 60 seconds). Nothing was sent. Get a new quote
and sign that.",
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#quote_expired",
    "details": {
      "freshQuote": {
        "route": "evm-btc-escrow",
        "liquidityAvailable": true,
        "from": {
          "chain": 8453,
          "chainName": "Base",
          "token": "0x0000000000000000000000000000000000000000",
          "symbol": "ETH",
          "decimals": 18,
          "caip2": "eip155:8453",
          "assetId": "eip155:8453/slip44:60"
        },
        "to": {
          "chain": "bitcoin",
          "chainName": "Bitcoin",
          "token": "BTC",
          "symbol": "BTC",
          "decimals": 8,
          "caip2": "bip122:000000000019d6689c085ae165831e93",
          "assetId": "bip122:000000000019d6689c085ae165831e93/slip44:0"
        },
        "fromAmount": "4000000000000000",
        "toAmount": "10882",
        "toAmountMin": "10882",
        "formatted": {
          "send": "0.004 ETH on Base",
          "receive": "0.00010882 BTC",
          "pair": "ETH on Base → BTC"
        },
        "expiresAt": 1794535605,
        "expiresAtIso": "2026-11-13T02:06:45.000Z",
        "limits": {
          "minSats": 546
        },
        "quoteId": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
        "order": {
          "route": "evm-btc-escrow",
          "orderId": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
          "chainId": 8453,
```

```
          "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
          "intent": {
            "side": 0,
            "maker": "0xC10B40524740Bc968e67D90c2088D5EF30635f2e",
            "asset": "0x0000000000000000000000000000000000000000",
            "scriptHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
            "satAmount": "10882",
            "designatedFiller": "0x0000000000000000000000000000000000000000",
            "pricingMode": 0,
            "limitPrice": "0",
            "startPrice": "0",
            "endPrice": "0",
            "decayStartTime": "0",
            "decayEndTime": "0",
            "slippageBps": 0,
            "oracleDiscountBps": 0,
            "bondBps": 0,
            "requiredConfirmations": 1,
            "settlementTimeoutBlocks": 0,
            "deadline": "1794542745",
            "nonce": "0",
            "oracleConfigHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
          },
          "status": "/status?
orderId=0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff&chainId=8453&route=evm-
btc-escrow",
          "sandbox": {
            "scenario": "success",
            "fromAddress": "0xC10B40524740Bc968e67D90c2088D5EF30635f2e",
            "toAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
            "fromToken": "0x0000000000000000000000000000000000000000",
            "toToken": "BTC",
            "fromAmount": "4000000000000000",
            "toAmount": "10882",
            "usd": 7.7
          },
          "quote": {
            "expiresAt": 1794535605,
            "id": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
            "token": "52f407983e777c911ca63f707a945fce1c42851629c7e12e72d4eb3fa9e2f0ec",
            "ask":
"from=ETH+on+base&to=BTC&amount=0.004&fromAddress=0xC10B40524740Bc968e67D90c2088D5EF30635f2e&toAddr
          }
        },
        "issues": {
          "balance": null,
          "allowance": null,
          "orderCap": null
```

```
        },
        "steps": [
          {
            "id": "intent",
            "kind": "signature",
            "description": "Sign the order (EIP-712). No gas. Sandbox: this signature authorizes
nothing.",
            "signatureKind": "eip712",
            "typedData": {
              "domain": {
                "name": "Maxy Sandbox",
                "version": "1",
                "chainId": 8453,
                "verifyingContract": "0x0000000000000000000000000000000000000000"
              },
              "types": {
                "MaxySandboxOrder": [
                  {
                    "name": "orderId",
                    "type": "string"
                  },
                  {
                    "name": "step",
                    "type": "string"
                  },
                  {
                    "name": "notice",
                    "type": "string"
                  }
                ]
              },
              "primaryType": "MaxySandboxOrder",
              "message": {
                "orderId": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
                "step": "intent",
                "notice": "Maxy sandbox. This moves nothing and cannot be spent."
              }
            },
            "hash": "0xdab6834d468daab77a35596f3f7dabd8a554e78a788ca0fbf5aa40d65c331a61"
          },
          {
            "id": "order",
            "kind": "post",
            "description": "Send the order and its signature; you get back the transaction to
send.",
            "method": "POST",
            "path": "/orders",
            "body": {
              "order": {
```

```
                "route": "evm-btc-escrow",
                "orderId": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
                "chainId": 8453,
                "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
                "intent": {
                  "side": 0,
                  "maker": "0xC10B40524740Bc968e67D90c2088D5EF30635f2e",
                  "asset": "0x0000000000000000000000000000000000000000",
                  "scriptHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
                  "satAmount": "10882",
                  "designatedFiller": "0x0000000000000000000000000000000000000000",
                  "pricingMode": 0,
                  "limitPrice": "0",
                  "startPrice": "0",
                  "endPrice": "0",
                  "decayStartTime": "0",
                  "decayEndTime": "0",
                  "slippageBps": 0,
                  "oracleDiscountBps": 0,
                  "bondBps": 0,
                  "requiredConfirmations": 1,
                  "settlementTimeoutBlocks": 0,
                  "deadline": "1794542745",
                  "nonce": "0",
                  "oracleConfigHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
                },
                "status": "/status?
orderId=0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff&chainId=8453&route=evm-
btc-escrow",
                "sandbox": {
                  "scenario": "success",
                  "fromAddress": "0xC10B40524740Bc968e67D90c2088D5EF30635f2e",
                  "toAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
                  "fromToken": "0x0000000000000000000000000000000000000000",
                  "toToken": "BTC",
                  "fromAmount": "4000000000000000",
                  "toAmount": "10882",
                  "usd": 7.7
                },
                "quote": {
                  "expiresAt": 1794535605,
                  "id": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
                  "token": "52f407983e777c911ca63f707a945fce1c42851629c7e12e72d4eb3fa9e2f0ec",
                  "ask":
"from=ETH+on+base&to=BTC&amount=0.004&fromAddress=0xC10B40524740Bc968e67D90c2088D5EF30635f2e&toAddr
                }
              },
```

```
              "signatures": {
                "intent": "<the intent signature>"
              }
            }
          },
          {
            "id": "escrow",
            "kind": "transaction",
            "description": "Sandbox: do not send the transaction. POST /orders/submit with its
simulatedHash.",
            "chainId": 8453,
            "transaction": null
          }
        ],
        "signing": {
          "kind": "eip712",
          "payloads": [
            {
              "id": "intent",
              "typedData": {
                "domain": {
                  "name": "Maxy Sandbox",
                  "version": "1",
                  "chainId": 8453,
                  "verifyingContract": "0x0000000000000000000000000000000000000000"
                },
                "types": {
                  "MaxySandboxOrder": [
                    {
                      "name": "orderId",
                      "type": "string"
                    },
                    {
                      "name": "step",
                      "type": "string"
                    },
                    {
                      "name": "notice",
                      "type": "string"
                    }
                  ]
                },
                "primaryType": "MaxySandboxOrder",
                "message": {
                  "orderId":
"0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
                  "step": "intent",
                  "notice": "Maxy sandbox. This moves nothing and cannot be spent."
                }
```

```
              },
              "hash": "0xdab6834d468daab77a35596f3f7dabd8a554e78a788ca0fbf5aa40d65c331a61"
            }
          ],
          "approval": null,
          "after": {
            "method": "POST",
            "path": "/orders"
          }
        },
        "sandbox": true
      }
    }
  },
  "requestId": "e7b47aa9-fcbe-4b03-9181-2e3717bd542b",
  "sandbox": true
}
```


`POST /orders` (HTTP 409): a real sandbox answer, 13 November 2026 (`x-request-id` e7b47aa9-fcbe-4b03-9181-2e3717bd542b).

Get a new quote and sign that. Nothing was signed on chain and nothing was sent.

3.5d The loop

curl

```
curl -s -D - -o quote.json -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" --data-urlencode
"from=ETH on base" -d to=BTC -d amount=0.004 \
  -d fromAddress="$WALLET" -d toAddress="$MY_BTC_ADDRESS" | grep -i '^date:'
jq .expiresAt quote.json
```

TypeScript

```
const skew = Date.parse(res.headers.get('date')!) - Date.now();  // the server's clock, not the
device's
const msLeft = () => quote.expiresAt * 1000 - (Date.now() + skew);
if (msLeft() < 5_000) quote = await getQuote();                  // renew 5 seconds before it
closes
```

Python

```
skew = parsedate_to_datetime(r.headers["date"]).timestamp() - time.time()  # the server's clock,
not the device's
if quote["expiresAt"] - (time.time() + skew) < 5:
    quote = get_quote()                                                    # renew 5 seconds
before it closes
```

3.5e Errors on this page

Code	HTTP	Cause	Fix
`quote_expired`	409	The quote's `expiresAt` passed, or the intent's deadline	Get a new quote
`quote_mismatch`	409	The escrow would lock more than the quote and its price band allow	Get a new quote

Next: 3.6 Security model.

Last verified: 13 November 2026, against the sandbox.

3.6 Security model

After this page you can say what your user signs, what the settler enforces, and how the user gets coins back.

Before you start: 1.1 What Maxy is. **Time:** 8 minutes.

3.6a What the user signs

Buying BTC, the user signs one EIP-712 message, the intent. The wallet shows each field; this is what each one binds:

Field	What it binds
`maker`	The paying wallet. Only it can open this escrow or take it back
`asset`	The coin being paid (the zero address for the chain's native coin)
`satAmount`	Exactly how many sats the user receives
`limitPrice`	The price per sat, in the coin's smallest unit, scaled by 10^18; the escrow is `satAmount × limitPrice`
`slippageBps`	How far the price may move before the escrow opens (default 50, 0.5%)
`bondBps`	The bond the solver must post when it commits, as a share of the escrow (default 100, 1%); if the solver commits and does not pay, the bond goes to the user (half of it to whoever sends the refund, when that is not the user)
`requiredConfirmations`	Bitcoin confirmations the payment needs before the solver can claim (default 1)
`scriptHash`	keccak256 of the Bitcoin output script that must receive the BTC: the user's own address, decoded locally
`deadline`, `nonce`	When the order lapses, and its uniqueness on the settler

The intent hash is the order id, on chain and in the API. The SDK computes it itself and compares it with the settler's own `hashIntent` before the wallet is asked to sign. A mismatch stops the order with nothing signed.

Paying in USDC on the gasless route, the wallet signs at most two messages: an EIP-2612 permit for exactly this order's USDC, and a Permit2 witness bound to the intent hash. The permit is skipped when the Permit2 allowance already covers the order. The solver submits the escrow and pays the gas.

3.6b What the settler enforces

The settler is one contract per chain, at an address compiled into the SDK and listed on the contracts page. A server response never picks the contract a user signs for.

An escrow has exactly four exits:

Exit	To	When	Contract function
Settle	the solver	it proves to the verifier that it paid the BTC to `scriptHash`, with the required confirmations	`fillIntentWithFeeToken`, against a Bitcoin proof
Cancel	the user	no solver committed: the user may cancel at any time, anyone else after the escrow deadline (2 hours after signing)	`cancelBuyEscrow`, `cancelNativeBuyEscrow`
Slash	the user, plus the solver's bond (half of it to the caller, when that is not the user)	a solver committed and did not prove payment before the Bitcoin deadline (144 blocks after the commit)	`slashExpired`
Refund	the user	the Bitcoin deadline passed and there was no bond	`refundExpired`

Cancel, slash and refund have no caller check once their deadline has passed, and pausing the settler never blocks them: no escrow can be trapped.

`swap.refund({ chainId, intentHash, evm })` picks the right call and sends it from the user's wallet. `swap.planRefund(chainId, intentHash, caller)` says which call applies and when it opens. The SDK waits for the escrow deadline before it cancels, even though the contract lets the user cancel earlier. The refunds page has the timings and the words to show.

3.6c What the verifier proves

The verifier is an on-chain Bitcoin light client. Bitcoin block headers are submitted to it, and it checks their proof of work itself. A payment proof is the raw Bitcoin transaction, its Merkle branch and the headers above it. The settler releases an escrow only when the verifier accepts such a proof for an output paying `scriptHash`. No Maxy server can mark a payment as made.

3.6d What Maxy can and cannot do

Maxy can	Maxy cannot
Quote prices and decline to fill an order	Move a user's escrow anywhere but back to the user, except to the solver against a Bitcoin proof
Upgrade the settler, through a timelock (contracts)	Stop a user from taking an expired escrow back: pausing never blocks refunds
Revoke your API key or lower its cap	Sign for the user, or change what the user signed
See the addresses and amounts of orders placed with your key	See or hold any private key

---

3.6e Selling BTC

Selling runs the other way round. The solver locks the payout in the settler for the user first. The SDK reads that escrow on chain and only then shows a Bitcoin deposit address. The address is decoded from the script the settler stores, never from server text. A proof of the user's BTC payment releases the payout to the user. Anyone may submit it, and `swap.claimBtcToEvm` submits it from the user's own wallet. If the user never pays, the solver takes its escrow back after the Bitcoin deadline.

3.6f The checks the SDK runs before a wallet prompt

The SDK's rule: a server response may choose among things the client already trusts, never introduce something new to sign or pay. Contracts, chain ids, EIP-712 domains and solver wallets are compiled in. Before every prompt it checks the intent hash twice and reads the settler nonce on chain. It also bounds the deposit and price, decodes the payout script locally, and simulates the escrow transaction. The full numbered list is security.

Integrating over HTTP without the SDK, run the same checks. Compare `steps[].typedData` with what you expect, and compute the EIP-712 hash yourself (`steps[].hash` is given for comparison). Simulate the escrow transaction before your user sends it (`POST /orders` returns `simulation`).

3.6g Errors on this page

None: this page has no call. The SDK's refusals before a wallet prompt are `verification_failed` , `quote_mismatch` and `tx_reverted` , on error codes.

Next: 3.7 Environments and keys.

Last verified: 5 November 2026, against the SDK source and the API it runs.

3.7 Environments and keys

After this page you know where your key may live, what it can do, and which environment it runs in.

Before you start: 3.2 Pricing anatomy. **Time:** 5 minutes.

3.7a Your API key

- **What it is.** `mxk_live_` followed by 32 letters and digits, issued by Maxy for your integration. It is shown once; Maxy keeps only a hash of it. A test key is the same with `mxk_test_`: only the prefix differs, and only the sandbox answers it.
- **How to send it.** `x-maxy-key: mxk_live_…` or `Authorization: Bearer mxk_live_…`.
- **What needs it.** Every route: chains, tokens, pairs, prices, quotes, orders, order status, address checks, webhooks, your order list and `GET /key`. Only `/health`, the status page and `POST /sandbox/keys` answer without one.
- **What it can do.** Ask for quotes and place orders that your users sign, register webhooks, and read the orders placed with it. It cannot move funds, sign anything, or read any wallet's records.
- **Its order size cap.** $100 per order unless agreed otherwise. `GET /key` shows it; ask to raise it.

Key	Where it may live	Why
Your `mxk_live_` key	Your server (recommended); a browser bundle only for a frontend that calls nothing but prices and quotes	A key in a page can be copied: someone could spend your rate limit, read your order list and register webhooks. It still cannot move anyone's funds

Rotating a key. `POST /keys/rotate` answers a new key with the same cap, once. Deploy it, then `DELETE /keys/{prefix}` revokes the old one. Both work until then. A revoked key is refused with `403 key_revoked` within about a minute; an unknown key with `401 invalid_api_key`.

3.7b Publishable keys

For a page, use a publishable key, bound to your site's origins: `POST /keys` with `{ "kind": "publishable", "origins": [...] }`. It reads, quotes and places the user's own signed order, and nothing else: every order still needs the user's wallet.

	Secret key	Publishable key
Prefix	`mxk_live_`, `mxk_test_`	`mxk_pk_live_`, `mxk_pk_test_`
Lives in	Your server	Your page

	Secret key	Publishable key
Can	Everything below	Prices, quotes, order status, address checks
Cannot		Place orders, register webhooks, list orders: `403 scope_denied`
Bound to	Nothing	The origins Maxy issued it for: elsewhere `403 origin_not_allowed`

An order still comes from your server with the secret key, as on 4.4 DeFi or DEX frontend. An origin check stops another site using your key in a browser; it cannot stop a server pretending to be your site, so a publishable key is never worth more than its rate limit. `GET /key` and `GET /limits` show a key's `kind`, `scopes` and `origins`.

Response

```
{
  "key": "mxk_pk_test_rOOYPo…",
  "kind": "publishable",
  "scopes": [
    "read",
    "quote",
    "place"
  ],
  "origins": [
    "http://localhost:3000"
  ],
  "orderCapUsd": 100,
  "sandbox": true
}
```

`GET /key`: a real sandbox answer, 13 November 2026 (`x-request-id` b13923d1-a49d-4e3d-9ba6-a55594762eae).

Response

```
{
  "error": {
    "code": "scope_denied",
    "message": "A publishable key reads, quotes and places the user's own signed order. GET
/orders takes your secret key, from your server.",
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#scope_denied"
  },
  "requestId": "e54cf4c0-882e-4379-8307-6cd2daf1315f"
}
```

`GET /orders` (HTTP 403): a real sandbox answer, 13 November 2026 (`x-request-id` e54cf4c0-882e-4379-8307-6cd2daf1315f).

Response

```
{
  "error": {
    "code": "origin_not_allowed",
    "message": "This publishable key is not issued for https://other.example. Ask Maxy to add
the origin, or use another key.",
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#origin_not_allowed"
  },
  "requestId": "a3b73484-0312-463c-8ef5-36e66c908837"
}
```

`GET /price` (HTTP 403): a real sandbox answer, 13 November 2026 (`x-request-id` a3b73484-0312-463c-8ef5-36e66c908837).

The SDK throws at construction if a secret key is used in a browser page.

3.7c Environments

The key chooses the environment, on one base URL. There is no testnet: a test key runs the sandbox. 8.3 Test scenarios lists how to produce each case in both.

	Test key	Live key
Prefix	`mxk_test_`	`mxk_live_`
Base URL	the same	the same
Orders	Sandbox table, walk on a timer	Real swaps on mainnet
Rate limit	120 calls a minute	600 calls a minute
Marked on every answer	`"sandbox": true`, `X-Maxy-Sandbox: true`	no mark

3.7d See your key

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/key"
```

TypeScript

```
const key = await fetch(`${BASE}/key`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
key = requests.get(f"{BASE}/key", headers={"x-maxy-key": KEY}).json()
```

Response

```
{
  "key": "mxk_test_8GOVco…",
  "kind": "secret",
  "scopes": [
    "read",
    "quote",
    "place",
    "orders",
    "webhooks",
    "account"
  ],
  "origins": [],
  "orderCapUsd": 100,
  "sandbox": true
}
```

`GET /key`: a real sandbox answer, 13 November 2026 (`x-request-id` 104dbd34-c9fd-41e7-99ff-daa1f12c9c3b).

3.7e Errors on this page

Code	HTTP	Cause	Fix
`auth_required`	401	The route needs a key and none was sent	Send `x-maxy-key`
`invalid_api_key`	401	The key is malformed or unknown	Check the key; keys look like `mxk_live_` and 32 letters and digits
`key_revoked`	403	The key was revoked	Use your new key

Next: 3.8 Rate limits, retries and idempotency.

Last verified: 13 November 2026, against the sandbox.

3.8 Rate limits, retries and idempotency

After this page you know how many calls your key allows, what a 429 looks like, and which calls are safe to repeat.

Before you start: 3.7 Environments and keys. **Time:** 5 minutes.

3.8a Rate limits

Caller	Limit	Counted
With a key	600 calls a minute	per key, wherever the calls come from, so your backend can serve every user from one address
Without a key (`/health`, the status page, getting a test key)	120 calls a minute	per IP address
Price reads, `GET /price` and each `GET /prices/stream` connection	Ten times the above: 6,000 and 1,200 a minute	apart from every other call; the same question within 3 seconds is one read of the solver
Order-creating calls	also limited per paying wallet	per wallet

Every answer from the API carries these headers (a key Maxy does not know is refused before the API, with `x-request-id` only):

HTTP

```
RateLimit-Limit: 600
RateLimit-Remaining: 599
RateLimit-Reset: 60
```

The window rolls over the last 60 seconds; `RateLimit-Reset` and `Retry-After` say 60. Over the limit the API answers `429` with `Retry-After` and does not act on the call:

Response

```
{
  "error": {
    "code": "rate_limited",
    "message": "Too many requests: 600 a minute. Try again in 60 s, or ask Maxy for a higher
limit.",
    "retryable": true,
    "docs": "https://maxy.co/docs/sdk-errors"
  },
  "requestId": "…"
}
```

Wait `Retry-After` seconds, then retry. The SDK's client does this for you, twice, and queues a burst instead of collecting 429s (the SDK client); after that it raises `RateLimitError` with `retryAfterMs`. Ask Maxy for a higher limit.

A sandbox answer, forced with `scenario=rate_limited`:

Response

```
{
  "error": {
    "code": "rate_limited",
    "message": "Too many requests (sandbox scenario rate_limited). Try again in 5 s.",
    "retryable": true,
    "docs": "https://maxy.co/docs/sdk-errors#rate_limited"
  },
  "requestId": "275a5744-a161-41e8-9bd6-8dc8327834a3",
  "sandbox": true
}
```

`GET /quote` (HTTP 429): a real sandbox answer, 13 November 2026 (`x-request-id` 275a5744-a161-41e8-9bd6-8dc8327834a3).

curl

```
curl -s -D - -o /dev/null -H "x-maxy-key: $MAXY_API_KEY" "$BASE/key" | grep -i -E
'ratelimit|retry-after'
```

TypeScript

```
const res = await fetch(`${BASE}/key`, { headers: { 'x-maxy-key': KEY } });
if (res.status === 429) await new Promise((r) => setTimeout(r, Number(res.headers.get('retry-
after')) * 1000));
```

Python

```
r = requests.get(f"{BASE}/key", headers={"x-maxy-key": KEY})
if r.status_code == 429:
    time.sleep(int(r.headers["retry-after"]))
```

3.8b Retries and idempotency

Every call except `POST /webhooks` can be retried as is. The order id is what makes a retry safe, so the API stores no idempotency keys: an `Idempotency-Key` header is accepted and has no further effect. A `POST /orders` after the quote's 60 seconds answers `409 quote_expired`: get a new quote.

The SDK's client retries by itself: reads, and any call given an idempotency key, twice, on 429, 502, 503, 504 or a dropped connection. `orders.create` and `orders.submit` set theirs from the order id and the transaction hash.

Call	Safe to retry?	Why
Any `GET`	yes	It changes nothing
`POST /orders` (escrow route)	yes	It sends nothing. The same signed order returns the same `orderId` and the same transaction; the settler accepts one escrow per intent
`POST /orders` (gasless route)	yes	The order id is the intent hash; the solver opens one escrow per intent, so a second submit cannot open a second one
`POST /orders/submit`	yes	It tells the solver about an escrow that is already on chain; the solver also watches the chain
`POST /webhooks`	creates a second endpoint	Check `GET /webhooks` first

Store the `orderId` the moment you have it. With it, `GET /status?orderId=` finds the order from any process; with only a transaction hash, `GET /status?txHash=` does.

Every answer carries `x-request-id` (also `requestId` in error bodies). Send your own in the `x-request-id` header (8 to 64 letters, digits, `.`, `_`, `-`) and the answer keeps it, so one id runs through your logs and Maxy's. Include it when you write to Maxy about a call.

The same signed escrow order, posted twice, answers the same `orderId`:

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "expiresAt": 1794542691,
  "transaction": "…",
  "simulation": "…",
  "next": "…",
  "expiresAtIso": "2026-11-13T04:04:51.000Z",
  "refundAt": 1794542691,
  "sandbox": true
}
```


`POST /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` e944a1bf-677f-4790-8840-ce398d3b4af6), with long fields folded.

3.8c Errors on this page

Code	HTTP	Cause	Fix
`rate_limited`	429	Over the per-key or per-IP limit	Wait `Retry-After` seconds
`quote_expired`	409	A retried `POST /orders` after the quote's 60 seconds	Get a new quote
`conflict`	409	A sixth webhook endpoint from a retried `POST /webhooks`	`GET /webhooks`, then delete one

Next: 3.9 Versioning and deprecation.

Last verified: 13 November 2026, against the sandbox.

3.9 Versioning and deprecation

After this page you know what can change under you, how much notice you get, and where to read about it.

Before you start: nothing. **Time:** 3 minutes.

3.9a The HTTP API

- Routes carry no version: `$BASE/price`, `$BASE/orders/{id}`.
- The API only gains fields and routes. Nothing is removed or renamed, so ignore fields you do not know.
- A breaking change is announced to every key holder at least 90 days ahead, with a migration guide. The old behaviour keeps working for at least six months.
- Every change is in the changelog.

3.9b The deprecation window

Change	Notice	Old version
A new field or route	None needed: ignore what you do not know	Stays
A breaking change	Announced to every key holder at least 90 days ahead	Keeps working for at least six months

A breaking change gets a migration guide before the notice goes out.

3.9c The SDK

The SDK follows semantic versioning. Below 1.0 a breaking change raises the minor version. The changelog lists what the versioning covers. `@maxy-labs/sdk` is delivered as a package file in your partner kit, so a new version reaches you only when you install it.

3.9d The route list

`GET $BASE` lists every route and whether it takes a key. New routes appear there first.

curl

```
curl -s "$BASE" | jq '.endpoints[:2]'
```

TypeScript

```
const { endpoints } = await fetch(BASE).then((r) => r.json());
```

Python

```
endpoints = requests.get(BASE).json()["endpoints"]
```

Response


```
{
  "name": "Maxy API",
  "sdk": "0.18.0",
  "baseUrl": "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api",
  "try": "curl -s -X POST https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-
api/sandbox/keys",
  "docs": "https://maxy.co/docs/first-order",
  "openapi": "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/openapi.json",
  "auth": "Every route takes your key in x-maxy-key (or Authorization: Bearer mxk_live_…), apart
from /health, /openapi.json, the status page (/system/*) and POST /sandbox/keys, which gives you
a test key.",
  "endpoints": [
    {
      "route": "GET /chains",
      "key": true,
      "what": "Chains Maxy swaps on"
    },
    {
      "route": "GET /tokens",
      "key": true,
      "what": "Tokens per chain (?chain=base)"
    },
    "…"
  ]
}
```


`GET` : a real sandbox answer, 13 November 2026 (`x-request-id` 8277cf47-b037-45e7-9d73-56c7bcc5b9d0), with long fields folded.

3.9e Errors on this page

This page's call needs no key and has no error of its own.

Next: 4.1 Paths at a glance.

Last verified: 13 November 2026, against the sandbox.

4.1 Paths at a glance

After this page you know how Maxy fits the system you already run, which endpoints your path uses, and which decisions are yours to make.

Before you start: 1.4 Which path is mine. **Time:** 5 minutes.

Path	Who	Endpoints it uses	Signs where
4.2 Aggregator or router adapter	Route aggregators, swap APIs	`pairs`, `price`, `quote`, `status?txHash=`	the user's wallet, through your executor
4.3 Wallet, exchange or on-ramp embed	Exchanges, consumer wallets, card on-ramps	`price`, `quote`, `orders`, `orders/submit`, `webhooks`, `orders` (ledger)	the user's wallet in your app
4.4 DeFi or DEX frontend	A "Buy BTC" button on an existing dapp	`price` (from the browser); `quote`, `orders`, `orders/submit` (through your server)	the connected wallet (wagmi, an injected provider)
4.5 Backend-only	Custodial platforms, treasury tools	`quote`, `orders`, `orders/submit`, `webhooks`, `orders` (ledger)	your servers, with your own key management

Every path places the same order: the user's coins go into the settler on their own chain, and the BTC goes to a Bitcoin address the user controls. Maxy holds nothing in between (security model).

4.1a Fields at a glance

You need	SDK	HTTP API
Amount in	`quote.sendAmount`, `quote.formatted.send`	`fromAmount`, `formatted.send`
Amount out	`quote.receiveAmount`, `quote.formatted.receive`	`toAmount`, `formatted.receive`
Quote expiry	`quote.expiresAt` (Unix seconds, 60 s)	`expiresAt`
Order id	`order.orderId` (the settler intent hash for buys)	`quoteId`, then `orderId`
Transactions	`order.txHashes`, `status.txs`, `status.btcTxId`	`status.txs`, `status.btcTxId`
Selling deposit	`order.deposit.address`, `order.deposit.sats` (read from the contract)	`GET /orders/{id}/deposit`: `address`, `sats` (read from the contract)
Selling claim	`swap.claimBtcToEvm()`	`GET /orders/{id}/claim`: the transaction for any wallet to send
SOL to BTC deposit	`swap.createOrder()`	`POST /orders`, then `POST /orders/{id}/steps` with the signed transaction

You need	SDK	HTTP API
Refund	`swap.planRefund()`, `swap.refund()`	`GET /orders/{id}/refund`: the transaction for the user's wallet to send
Selling, two legs	`swap.bridgeBtcToEvmPayout()`	`GET /orders/{id}/bridge`: the transaction for the user's wallet to send
BTC to SOL	`swap.createOrder()`, `swap.settleBtcToSol()`	`POST /orders`, then `POST /orders/{id}/steps`, stage by stage
EVM to EVM	`swap.createOrder()`	`POST /orders`, then `POST /orders/submit` with the hash
Exact BTC	`swap.quote({ receive: '1.31' })`	`receive=1.31` or `toAmount` on `/price` and `/quote`
Every pair	`swap.pairs()`	`GET /pairs`

Each guide lists the errors its calls can return; every code is in HTTP API error codes.

4.1b Errors on this page

Each path lists its own. Every code is on error codes.

Next: 4.2 Aggregator or router adapter.

Last verified: 5 November 2026, against the SDK source and the API it runs.

4.2 Aggregator or router adapter

After this page you can wrap Maxy as one more source in your router: discovery, ranking, skipping a size, execution and tracking.

Before you start: Keys and 3.2 Pricing anatomy. **Time:** 15 minutes.

4.2a How it fits

Maxy is one more source in your route list: a native BTC leg your router can end on.

```
your router ── GET /pairs (cache 60 s) ──────────▶ Maxy: which pairs exist
            ── GET /price?…&amount=…  ──────────▶ Maxy: what each size gets (with your key)
user picks the Maxy route
            ── GET /quote?…&fromAddress&toAddress ▶ Maxy: quoteId, expiresAt, steps
your executor: the user's wallet signs the steps and sends the escrow transaction ──▶ settler
your tracker ── GET /status?txHash=… ──────────▶ Maxy: state, btcTxId
```

The first call.

curl

```
curl -s "$BASE/pairs"
```

TypeScript

```
const { pairs } = await fetch(`${BASE}/pairs`).then((r) => r.json());
```

Python

```
pairs = requests.get(f"{BASE}/pairs").json()["pairs"]
```

Response


```
{
  "pairs": [
    {
      "from": {
        "chain": 8453,
        "token": "0x0000000000000000000000000000000000000000",
        "symbol": "ETH",
        "decimals": 18,
        "chainName": "Base"
      },
      "to": {
        "chain": "bitcoin",
        "token": "BTC",
        "symbol": "BTC",
        "decimals": 8,
        "chainName": "Bitcoin"
      },
      "label": "ETH on Base → BTC",
      "fromAssetId": "eip155:8453/slip44:60",
      "toAssetId": "bip122:000000000019d6689c085ae165831e93/slip44:0",
      "route": "evm-btc-escrow",
      "settler": "0xeC5a05b28dd848F628e19CB26Df32f053e64ab86",
      "minSats": 546,
      "requiredConfirmations": 1,
      "confirmationsBySize": false
    },
    "…"
  ],
  "sandbox": true
}
```


`GET /pairs`: a real sandbox answer, 13 November 2026 (`x-request-id` 0a46da56-61c4-4911-8f16-de025ca1482c), with long fields folded.

Decisions that are yours.

- **Discovery.** Read `GET /pairs` and cache it for up to 5 minutes (the API sends `max-age=300`).
- **Ranking.** Rank on `toAmount`. It is executable, so `toAmountMin` equals it.
- **Size.** Prices answer at any size. Before you show Maxy as the best route, get the quote: when it carries `issues.orderCap`, skip Maxy for that size instead of failing at execution.
- **Funds first.** A USDC quote names its steps only for a wallet that already holds the USDC. When your router's own leg delivers the USDC, quote Maxy again after it lands.
- **The order id.** Keep `quoteId` (it becomes the `orderId`) or rely on the transaction hash: `GET /status?txHash=` finds the order from any of its transactions, EVM or Bitcoin.

- **Timing.** A buy's BTC is sent within seconds of the escrow landing. Give your route an `executionDuration` of 15 seconds after the escrow, and 1,800 for sells (Bitcoin confirmations).

Map a Maxy quote to the route shape aggregators share:

TypeScript

```
import type { SwapQuote } from '@maxy-labs/sdk';

const asRoute = (q: SwapQuote) => ({
  id: `maxy:${q.route}:${q.sendAmount}:${q.expiresAt}`,
  fromAmount: q.sendAmount.toString(),
  toAmount: q.receiveAmount.toString(),
  toAmountMin: q.receiveAmount.toString(), // executable: the escrow's price band covers price
moves
  executionDuration: q.to.chain === 'bitcoin' ? 15 : 1800,
  steps: [{ type: 'cross', tool: 'maxy', estimate: { toAmount: q.receiveAmount.toString() } }],
});
```

Run a `tool: 'maxy'` step with `maxy.swap.createOrder(quote, { evm, btcAddress })`, or over HTTP with the steps the quote returns (buy BTC from an EVM chain).

A Maxy quote reads like a cross-chain entry in a swap API's response:

Swap-quote field	Maxy (`GET /quote`)
`sellAmount`	`fromAmount`
`buyAmount`	`toAmount` (what arrives)
`minBuyAmount`	`toAmountMin` (equal to `toAmount`)
`liquidityAvailable`	`true` when there is an amount; `false` (HTTP 200, no amounts, a `message`) when Maxy cannot fill this order now: skip Maxy for that size. Over your key's cap, `issues.orderCap` is set and `steps` is null
`issues.balance` , `issues.allowance`	`issues.balance` , `issues.allowance`
`transaction`	the `transaction` step, or `POST /orders` ' `transaction`
`route.fills[].source`	`'maxy'`

A native BTC leg after your own route. Route any token to USDC on Base in the user's wallet. Then Maxy takes what arrived to BTC, gasless:

TypeScript

```
// fragment: maxy, evm and btcAddress as in the quickstart; routerStatus is your router's own
status answer
// 1. Your router: any token, any chain -> USDC on Base, to the user's wallet. Your status call
says what arrived.
const arrived: bigint = BigInt(routerStatus.receiving.amount); // 6-decimal units

// 2. Maxy: that USDC -> BTC at the user's Bitcoin address. The user signs and pays no gas.
const { status } = await maxy.swap.execute({ from: 'USDC on base', to: 'BTC', amount: arrived,
evm, btcAddress });
```

A wallet SDK with provider plugins. A Maxy plugin is one `swap({ route })`. It hands the route to the Maxy SDK, with your SDK's wallet as the signer. Asset notation like `BASE.ETH` and `BTC.BTC` is read as is:

TypeScript

```
import { createMaxyClient, evmSignerFromEip1193 } from '@maxy-labs/sdk';

const maxy = createMaxyClient({ apiKey: process.env.MAXY_API_KEY });

export const maxyPlugin = {
  name: 'maxy',
  async swap({ route, wallet }) {
    const { order } = await maxy.swap.execute({
      from: route.sellAsset,        // 'BASE.ETH'
      to: route.buyAsset,           // 'BTC.BTC'
      amount: route.sellAmount,     // '0.05' (whole coins) or a bigint in wei
      evm: evmSignerFromEip1193(wallet.provider),
      btcAddress: route.destinationAddress,
    });
    return order.txHashes[0] ?? order.orderId; // the source transaction hash; gasless orders
have none
  },
};
```

Where the rest sits. Identity checks and user terms stay with you and your front end. Your router holds the order id, or finds it again from the transaction hash.

The reference implementation. `examples/integrate/paths/adapter.ts` runs against the sandbox: discovery, a route, a skipped size and tracking by hash. Your partner kit carries it (runnable samples).

Go-live for this path: the checklist, plus: an over-cap size is skipped, not failed; `GET /status?txHash=` resolves your own test orders.

What a router also uses. `GET /pairs` gives each pair's route, settler, smallest order and confirmations. `POST /quotes/batch` evaluates up to 20 sizes or chains in one call, with an error per item. `GET /health` has a `degraded` flag: skip Maxy while it reads true, instead of timing out on a quote. Price reads have ten times the limit of other calls, and `GET /prices/stream` saves polling.

4.2b Errors on this page

Code	HTTP	Cause	Fix
`order_cap_exceeded`	422	Above your key's cap; quotes flag it in `issues.orderCap`	Order less
`quote_expired`	409	Posted after the quote's 60 seconds	New quote, sign again
`invalid_api_key`, `auth_required`	401	A key problem	Check `x-maxy-key`
`rate_limited`	429	Over your rate limit	Wait `Retry-After` seconds

The whole list is in error codes.

Next: 4.3 Wallet, exchange or on-ramp embed, or the 12.1 Source adapter.

Last verified: 13 November 2026, against the sandbox.

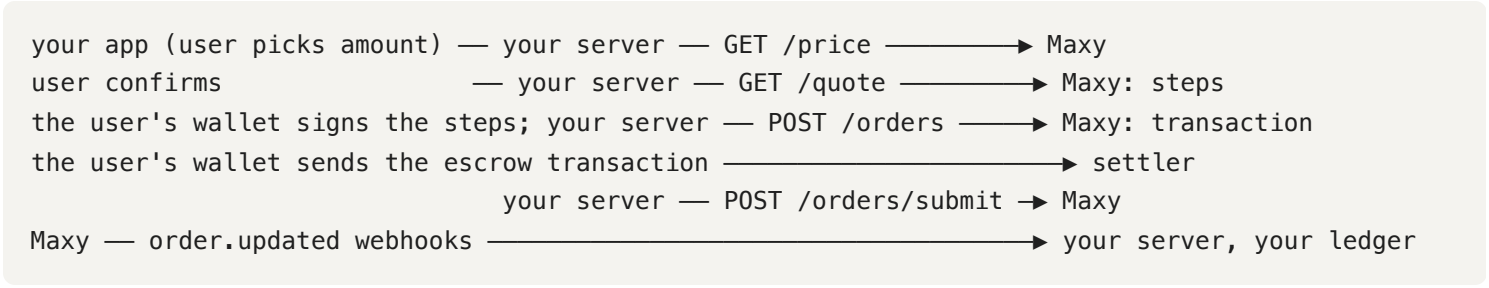
4.3 Wallet, exchange or on-ramp embed

After this page you can put a BTC buy inside your product, paid from the wallet your user already has.

Before you start: an API key and your own server. **Time:** 15 minutes.

4.3a How it fits

A BTC buy inside your product, paid from the wallet your user already has, landing in a Bitcoin address they own.



The first call. Your server asks for a quote; a test key runs it in the sandbox.

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" --data-urlencode "from=USDC on base" -d
to=BTC -d amount=10 \
  -d fromAddress="$WALLET" -d toAddress="$MY_BTC_ADDRESS"
```

TypeScript

```
const q = new URLSearchParams({ from: 'USDC on base', to: 'BTC', amount: '10', fromAddress,
toAddress: btcAddress });
const quote = await fetch(`${BASE}/quote?${q}`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
quote = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, params={
    "from": "USDC on base", "to": "BTC", "amount": "10", "fromAddress": wallet, "toAddress":
btc_address,
}).json()
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "10000000",
  "toAmount": "13669",
  "toAmountMin": "13669",
  "formatted": "…",
  "expiresAt": 1794535579,
  "expiresAtIso": "2026-11-13T02:06:19.000Z",
  "limits": "…",
  "quoteId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
  "order": "…",
  "issues": "…",
  "steps": [
    {
      "id": "witness",
      "kind": "signature",
      "description": "Sign the transfer bound to this order. No gas. Sandbox: this signature
authorizes nothing.",
      "signatureKind": "eip712",
      "typedData": "…",
      "hash": "0x1c3aa0811071c34e2f1b536f26ef9ed205ec6eda8598ecaa995d315f8629c707"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signatures. The solver opens the escrow and pays
the gas.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    }
  ],
  "signing": "…",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 686e7eae-9a18-4bba-a051-21694b9fd7d4), with long fields folded.

Decisions that are yours.

- **Identity checks.** Maxy performs none and keeps no user accounts. Your own onboarding rules apply as they do to any other trade in your product (security and compliance).

- **The BTC address.** Use a native SegWit address (`bc1q…`) the user controls: their wallet's own BTC account, or their BTC deposit address on your exchange. Check it with `GET /addresses/validate` before quoting.
- **The order id.** Store `orderId` against your own order the moment `POST /orders` returns, and key your ledger on it; webhooks carry it.
- **Your key.** It stays on your server; your app talks to your server.

A card on-ramp in front. Buy USDC on Base with the card, then let Maxy turn it into BTC: the BTC leg is gasless, so a user who has never held ETH can finish it.

TypeScript

```
// fragment: maxy, evm and btcAddress as in the quickstart; notifyUser is your own
// Your on-ramp's "completed" webhook (or a poll of its transaction):
async function onRampCompleted(tx: { cryptoAmount: string; walletAddress: string }) {
  const { status } = await maxy.swap.execute({
    from: 'USDC on base',
    to: 'BTC',
    amount: tx.cryptoAmount,   // '100.00', as the on-ramp reports it
    evm,                       // the wallet at tx.walletAddress
    btcAddress,
    onUpdate: (u) => u.step === 'status' && notifyUser(u.status.state),
  });
  return status.btcTxId;
}
```

Pass your own order id through the on-ramp's external-transaction field. When its webhook arrives, look up that order to find the user's BTC address.

The reference implementation. `examples/integrate/paths/webhook-receiver.mjs` is the ledger side: it verifies each delivery and processes each event `id` once. `sandbox.ts` is the order side (runnable samples).

Go-live for this path: the checklist, plus: every state in order lifecycle has your approved words. The refunds page is in your support team's runbook. Webhooks update your ledger once per event `id`.

4.3b Errors on this page

Code	HTTP	Cause	Fix
`order_cap_exceeded`	422	Above your key's cap; quotes flag it in `issues.orderCap`	Order less
`quote_expired`	409	Posted after the quote's 60 seconds	New quote, sign again

Code	HTTP	Cause	Fix
`invalid_api_key`, `auth_required`	401	A key problem	Check `x-maxy-key`
`rate_limited`	429	Over your rate limit	Wait `Retry-After` seconds

The whole list is in error codes.

Next: 4.4 DeFi or DEX frontend, or the 12.2 Product embed.

Last verified: 13 November 2026, against the sandbox.

4.4 DeFi or DEX frontend

After this page you can add a Buy BTC button on the wallet connection your dapp already has, with your key on your server.

Before you start: an existing wagmi or injected wallet connection, and a place for one server route.
Time: 15 minutes.

4.4a How it fits

A "Buy BTC" button on the wallet connection your dapp already has. Your secret key stays on your server behind one small route; the browser only signs and sends what the quote lists. The page can read prices and quotes itself with a publishable key bound to your origin (keys), and `@maxy/react` has the hooks for it: React. What follows is the same thing with no package at all.

```
browser: GET /price (your publishable key, from your origins) while the user types
browser ── your /api/maxy/* ── GET /quote, POST /orders, POST /orders/submit (your key) ──▶ Maxy
browser: the connected wallet signs the quote's steps and sends the escrow transaction
```

TypeScript

```
// app/api/maxy/[...path]/route.ts (Next.js): forwards three calls to Maxy with your key. The
key never reaches the browser.
const MAXY = 'https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api';
const ALLOWED = ['GET /quote', 'POST /orders', 'POST /orders/submit'];

async function forward(req: Request) {
  const url = new URL(req.url);
  const path = url.pathname.replace(/^\/api\/maxy/, '');
  if (!ALLOWED.includes(`${req.method} ${path}`)) return Response.json({ error: 'not allowed' },
{ status: 404 });
  const res = await fetch(`${MAXY}${path}${url.search}`, {
    method: req.method,
    headers: { 'x-maxy-key': process.env.MAXY_API_KEY!, 'content-type': 'application/json' },
    body: req.method === 'POST' ? await req.text() : undefined,
  });
  return new Response(res.body, { status: res.status, headers: { 'content-type':
'application/json' } });
}
export { forward as GET, forward as POST };
```

TypeScript

```
// In the browser, on your existing wagmi config.
import { getAccount, sendTransaction, signTypedData, waitForTransactionReceipt } from
'@wagmi/core';
import { wagmiConfig } from './wagmi'; // your app's own config
import { hashTypedData } from 'viem';

const maxy = (path: string, body?: unknown) =>
  fetch(`/api/maxy${path}`, body === undefined ? undefined : { method: 'POST', headers: {
'content-type': 'application/json' }, body: JSON.stringify(body) })
    .then((r) => r.json());

export async function buyBtc(from: string, amount: string, btcAddress: string): Promise<string>
{
  const { address } = getAccount(wagmiConfig);
  const quote = await maxy(`/quote?${new URLSearchParams({ from, to: 'BTC', amount, fromAddress:
address!, toAddress: btcAddress })}`);
  if (quote.issues?.orderCap) throw new Error(quote.issues.orderCap.message); // "Order size is
capped for this API key. Try $100 or less."
  if (!quote.steps) throw new Error(quote.error?.message ?? 'Not enough funds for this order.');

  const signatures: Record<string, string> = {};
  for (const step of quote.steps) {
    if (step.kind === 'transaction' && step.transaction) { // only a token without an EIP-2612
permit: approve this order's amount
      const { from: _from, ...tx } = step.transaction;
      await waitForTransactionReceipt(wagmiConfig, { hash: await sendTransaction(wagmiConfig, {
...tx, value: BigInt(tx.value ?? 0) }) });
    }
    if (step.kind === 'signature') {
      if (hashTypedData(step.typedData) !== step.hash) throw new Error(`Step ${step.id} does not
match its hash.`);
      signatures[step.id] = await signTypedData(wagmiConfig, step.typedData);
    }
  }

  const placed = await maxy('/orders', { order: quote.order, signatures });
  if (placed.error) throw new Error(placed.error.message);
  if (placed.transaction) { // ETH, BNB, AVAX: the user's wallet sends the escrow
    const { from: _from, ...tx } = placed.transaction;
    const txHash = await sendTransaction(wagmiConfig, { ...tx, value: BigInt(tx.value) });
    await maxy('/orders/submit', { ...placed.next.body, txHash });
  }
  return placed.orderId; // follow it with GET /status?orderId=… (your key)
}
```

While the user types, the browser can call `GET /price` on Maxy directly with your publishable key (`mxk_pk_…`), which works only from the origins it was issued for.

curl

```
curl -s -H "x-maxy-key: $MAXY_PUBLISHABLE_KEY" -H "origin: https://your.app" "$BASE/price?from=USDC%20on%20base&to=BTC&amount=10"
```

TypeScript

```
const price = await fetch(`${BASE}/price?from=USDC%20on%20base&to=BTC&amount=10`, { headers: { 'x-maxy-key': PUBLISHABLE_KEY } }).then((r) => r.json());
```

Python

```
price = requests.get(f"{BASE}/price", params={"from": "USDC on base", "to": "BTC", "amount": "10"}, headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "10000000",
  "toAmount": "13667",
  "toAmountMin": "13667",
  "formatted": {
    "send": "10.00 USDC on Base",
    "receive": "0.00013667 BTC",
    "pair": "USDC on Base → BTC"
  },
  "expiresAt": 1794535530,
  "expiresAtIso": "2026-11-13T02:05:30.000Z",
  "limits": {
    "minSats": 546
  },
  "sandbox": true
}
```


`GET /price`: a real sandbox answer, 13 November 2026 (`x-request-id` a4e23a42-1909-467e-acc7-5a24f4d66c99), with long fields folded.

With the SDK in a Node context (a server, a script), `maxy.swap.execute` runs the same steps; map its progress to your stepper:

TypeScript

```
// fragment: request is { from, to, amount }; show is your own UI call
await maxy.swap.execute({ ...request, evm, btcAddress, onUpdate: (u) => {
  if (u.step === 'quote') show(`You get ${u.quote.formatted?.receive}`);
  if (u.step === 'order') show('Signed. Sending…');
  if (u.step === 'status') show(u.status.state);
} });
```

Maxy asks your wallet layer for one object with six methods, so your wallet stack stays in charge:

TypeScript

```
// fragment: account and wallet come from your own wallet layer
import type { EvmSigner } from '@maxy-labs/sdk';

const evm: EvmSigner = {
  getAddress: async () => account,
  getChainId: async () => wallet.chainId,
  switchChain: (id) => wallet.switchChain(id),
  signTypedData: (typed) => wallet.signTypedData(typed), // EIP-712
  signMessage: (msg) => wallet.signMessage(msg),
  sendTransaction: (tx) => wallet.send(tx), // { chainId, to, data, value } -> hash; do not wait
for the receipt
};
```

`evmSignerFromEip1193(provider)` and `evmSignerFromWalletClient(viemWalletClient)` build it for you (signers).

Where the rest sits. Identity checks, if your product has any, stay in your product. The browser holds the order id for the user's screen; your server may store it too.

The reference implementation. `examples/integrate/paths/next-route.ts` and `buy-btc.ts` are the two files above, ready to copy into a Next.js app with wagmi (runnable samples).

Go-live for this path: the checklist, plus: the API host is in your 7.3 Content Security Policy; the key is only on the server.

4.4b Errors on this page

Code	HTTP	Cause	Fix
`order_cap_exceeded`	422	Above your key's cap; quotes flag it in `issues.orderCap`	Order less
`quote_expired`	409	Posted after the quote's 60 seconds	New quote, sign again

Code	HTTP	Cause	Fix
`invalid_api_key`, `auth_required`	401	A key problem	Check `x-maxy-key`
`rate_limited`	429	Over your rate limit	Wait `Retry-After` seconds

The whole list is in error codes.

Next: 4.5 Backend-only, or the 12.3 Frontend button.

Last verified: 13 November 2026, against the sandbox.

4.5 Backend-only

After this page you can create and sign orders on your servers, with your own key management and no browser.

Before you start: an API key and your own signer (HSM, KMS or MPC). **Time:** 15 minutes.

4.5a How it fits

Orders created and signed on your servers, with your own key management, over plain HTTP. In Python, the `maxy` package has the same calls with retries and typed errors (Python). A custodial platform that drives each step itself can use `createSwapMachine` for the states and allowed moves.

```
your server ── GET /quote ──────────► Maxy: order + steps (typed data, hash)
your signer (HSM, KMS, MPC) signs the hash
your server ── POST /orders ────────► Maxy: the escrow transaction (simulated)
your signer sends it ──────────────────► settler
your server ── POST /orders/submit ─► Maxy
Maxy ── order.updated ──────────────────► your server
```

The first call. Your server asks for a quote; a test key runs it in the sandbox.

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" --data-urlencode "from=USDC on base" -d
to=BTC -d amount=10 \
  -d fromAddress="$WALLET" -d toAddress="$MY_BTC_ADDRESS"
```

TypeScript

```
const q = new URLSearchParams({ from: 'USDC on base', to: 'BTC', amount: '10', fromAddress,
toAddress: btcAddress });
const quote = await fetch(`${BASE}/quote?${q}`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
quote = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, params={
    "from": "USDC on base", "to": "BTC", "amount": "10", "fromAddress": wallet, "toAddress":
btc_address,
}).json()
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "10000000",
  "toAmount": "13669",
  "toAmountMin": "13669",
  "formatted": "…",
  "expiresAt": 1794535579,
  "expiresAtIso": "2026-11-13T02:06:19.000Z",
  "limits": "…",
  "quoteId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
  "order": "…",
  "issues": "…",
  "steps": [
    {
      "id": "witness",
      "kind": "signature",
      "description": "Sign the transfer bound to this order. No gas. Sandbox: this signature
authorizes nothing.",
      "signatureKind": "eip712",
      "typedData": "…",
      "hash": "0x1c3aa0811071c34e2f1b536f26ef9ed205ec6eda8598ecaa995d315f8629c707"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signatures. The solver opens the escrow and pays
the gas.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    }
  ],
  "signing": "…",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 686e7eae-9a18-4bba-a051-21694b9fd7d4), with long fields folded.

Decisions that are yours.

- **Signing.** Each signature step carries `typedData` and its EIP-712 `hash`. Sign the hash with your key system, or sign the typed data with a library and compare hashes.

- **Retries.** Every call except `POST /webhooks` can be retried as is; the order id is the idempotency key (retries).
- **Statements.** `GET /orders?from=&to=&format=csv` is your month-end export (reconcile).

The whole flow in curl, TypeScript and Python is buy BTC from an EVM chain.

Where the rest sits. Identity checks and custody rules are yours, because the signing key is yours. Your database holds the order id.

The reference implementation. `examples/integrate/paths/worker.py` runs a sandbox order end to end from a server: quote, sign with a local key, order, poll, export (runnable samples).

Go-live for this path: the checklist, plus: a retried `POST /orders` returns the same `orderId` in your tests; month-end reconciles from the CSV.

4.5b Errors on this page

Code	HTTP	Cause	Fix
`order_cap_exceeded`	422	Above your key's cap; quotes flag it in `issues.orderCap`	Order less
`quote_expired`	409	Posted after the quote's 60 seconds	New quote, sign again
`invalid_api_key`, `auth_required`	401	A key problem	Check `x-maxy-key`
`rate_limited`	429	Over your rate limit	Wait `Retry-After` seconds

The whole list is in error codes.

Next: 5.1 Buy BTC from an EVM chain, or the 12.4 Headless backend.

Last verified: 13 November 2026, against the sandbox.

4.6 The widget

After this page your site has a complete buy and sell screen, on the wallet your users already have.

Before you start: a publishable key bound to your site's origin (keys). **Time:** 10 minutes.

4.6a Add it

`@maxy/widget` is the whole UI: amount and coin, recipient entry with validation, paste, the quote countdown, a screen for every state, a receipt with the Bitcoin txid and explorer link, a refund screen for `refundable`, and the payment instructions and QR code for a sale. It never ships a wallet and never opens one of its own: it uses the page's injected wallet, or the wallet you pass.

TypeScript

```
import { MaxyWidget } from '@maxy/widget';

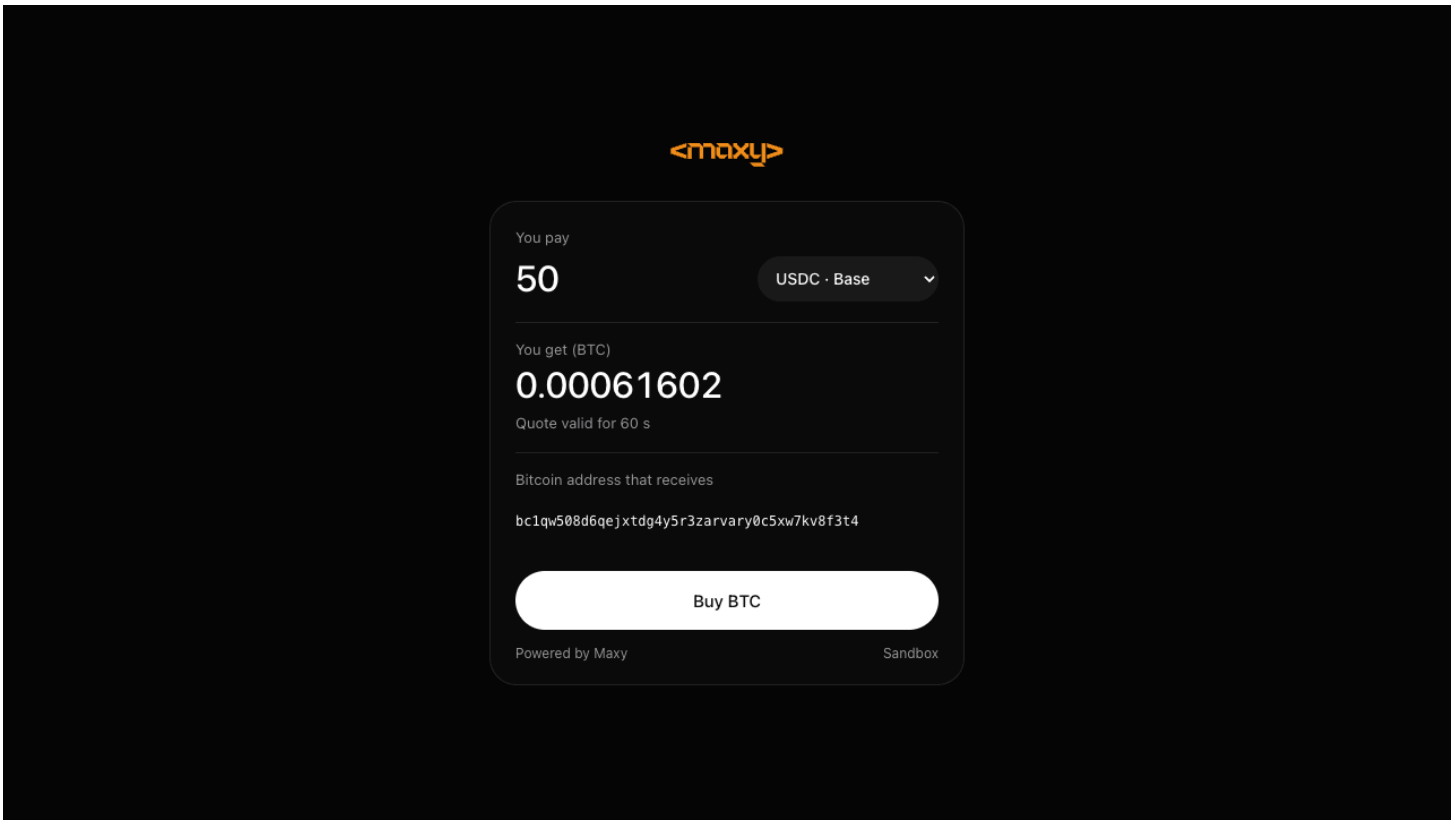
export function Checkout() {
  return (
    <MaxyWidget
      apiKey="mxk_pk_test_…"
      defaultFrom="USDC on base"
      defaultAmount="50"
      theme="auto"
      onEvent={(e) => console.log(e.type, e.orderId)}
    />
  );
}
```

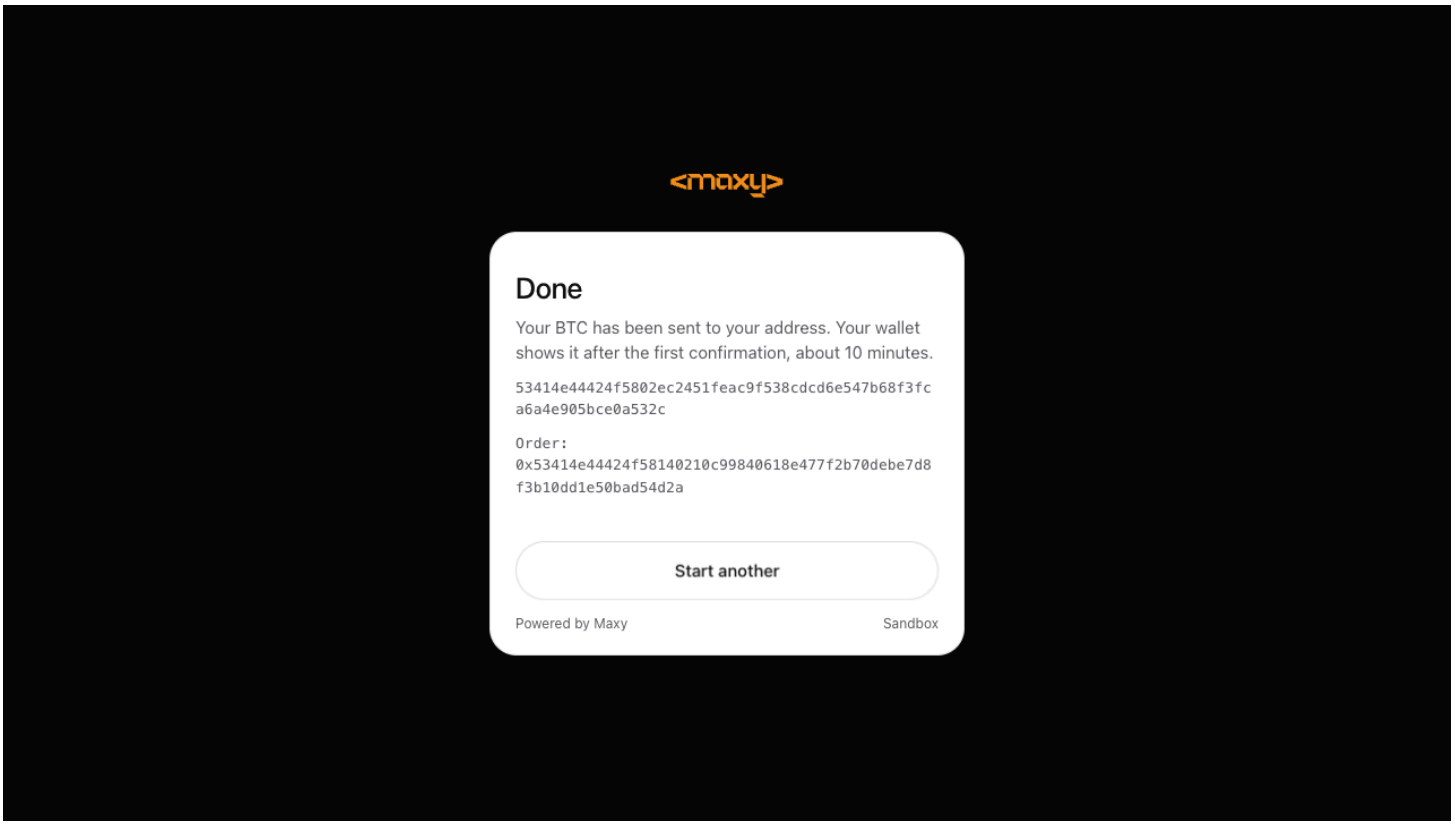
curl

```
# With no framework: one script tag. The file's Subresource Integrity hash ships beside it.
curl -s https://maxy-checkout.vercel.app/maxy-widget.min.js -o maxy-widget.min.js
```

Python

```
# The widget is a browser UI. From Python, sign a hosted checkout link instead: see the hosted
checkout page.
```





4.6b The props

Prop	What it does
`apiKey`	Your publishable key. A test one runs everything in the sandbox
`mode`	`both` (default), `buy` or `sell`
`chains`, `tokens`	Limit the coins: `[8453, "solana"]`, `["USDC", "ETH"]`
`defaultFrom`, `defaultAmount`	What it starts at
`recipient`, `recipientMode`	A fixed Bitcoin address, `locked`; `input` lets the user type or paste it
`theme`	`light`, `dark` or `auto`. Colours are CSS variables on `.mxw`: `--mxw-accent`, `--mxw-bg`, `--mxw-fg`
`locale`	`en`, `es`, `pt`, `fr`, `de`, `ja`, `ko`, `zh`, `tr`. Default: the browser's
`provider`	An EIP-1193 provider, such as a wagmi connector's. Default: the injected one
`evmWallet`	Or a ready wallet: `evmWallet(walletClient)` from `@maxy-labs/sdk/evm`
`solanaWallet`	A Solana wallet adapter's signer, to pay from Solana
`externalId`, `externalCustomerId`	Your ids: they come back on webhooks and statements
`onEvent`	`ready`, `state`, `completed`, `refundable`, `failed`, with `orderId` and `bitcoinTxid`

The script tag exposes `MaxyWidget.mount(element, props)`, with `update` and `unmount` on what it returns. The translations other than English have not had a native read.

4.6c How it behaves

The quote always shows, at any size, and the button is always clickable. Anything wrong is said after the click, with nothing sent: an amount over your key's cap, too small an amount, an address that cannot be paid. The order is done when the user's coins arrive: the receipt shows at that moment, and nothing waits for settlement. It shows no fee lines. It budgets 150 kB gzipped for the script-tag build, React included, and the build fails above that.

4.6d Content Security Policy

The widget adopts a constructed stylesheet, so `style-src` needs no `unsafe-inline` where the browser supports it. Your page needs:

```
connect-src https://whhxyutitykcsdpvdgcd.supabase.co
```

4.6e Errors on this page

Code	Cause	Fix
`origin_not_allowed`	The page's origin is not on the key's list	Add it with `POST /keys/{prefix}/origins`
`scope_denied`	A secret key was passed to a page's call	Use a publishable key

Next: 4.7 Hosted checkout.

Last verified: 5 November 2026, against the SDK source and the API it runs.

4.7 Hosted checkout

After this page you can send a user to a Maxy-hosted buy page with one signed link, and build no frontend at all.

Before you start: a secret key and a publishable key from the same account (keys). **Time:** 10 minutes.

4.7a Sign a link

Your server signs the link with your secret key: HMAC-SHA256 over the sorted query string. The page shows nothing if one character changes, so a user cannot alter the recipient. A link lasts an hour unless you say otherwise.

curl

```
# With the SDK, or by hand: sha256 of your key (a test key: of "maxy-sandbox:" + key) is the
HMAC key.
curl -s "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/checkout/verify?
$SIGNED_QUERY"
```

TypeScript

```
import { signCheckoutUrl } from '@maxy-labs/sdk/server';

const url = await signCheckoutUrl({
  apiKey: process.env.MAXY_PUBLISHABLE_KEY,
  from: 'USDC on base',
  defaultAmount: '50',
  recipient: 'bc1q…',
  externalId: 'cart-1234',
  redirectUrl: 'https://shop.example/thanks',
}, process.env.MAXY_API_KEY);
```

Python

```
import hashlib, hmac, os, time
from urllib.parse import quote

key = os.environ["MAXY_API_KEY"]
stored = hashlib.sha256(("maxy-sandbox:" + key if key.startswith("mxk_test_") else
key).encode()).hexdigest()
params = {"apiKey": os.environ["MAXY_PUBLISHABLE_KEY"], "kid": key[:15], "exp":
str(int(time.time()) + 3600), "from": "USDC on base", "defaultAmount": "50", "recipient":
"bc1q…"}
query = "&".join(f"{quote(k, safe='')}={quote(v, safe='')}" for k, v in sorted(params.items()))
url = f"https://maxy-checkout.vercel.app/?{query}&sig={hmac.new(stored.encode(), query.encode(),
hashlib.sha256).hexdigest()}"
```

Parameter	What it does
`apiKey`	Your publishable key
`kid`, `exp`, `sig`	Added by `signCheckoutUrl`: your secret key's prefix, the expiry, the signature
`from`, `defaultAmount`	What the page starts at. The user can change the amount
`recipient`	The Bitcoin address that receives: locked. Leave it out and the user types one
`externalId`, `externalCustomerId`	Your ids, on webhooks and statements
`redirectUrl`	Where the user goes when the order ends, with `orderId` and `state`. Must be on one of the key's origins
`theme`, `locale`	As the widget

4.7b Open it

Link to it, open it in a window, or frame it. The page's `frame-ancestors` are your publishable key's origins, so only your own site can frame it. Events reach your page by `postMessage`, sent only to those origins, and by webhook.

TypeScript

```
window.addEventListener('message', (e) => {
  if (e.origin !== 'https://maxy-checkout.vercel.app' || e.data?.source !== 'maxy-checkout')
return;
  if (e.data.event.type === 'completed') showThanks(e.data.event.orderId);
});
```

The page runs on `maxy-checkout.vercel.app`; `buy.maxy.co` is already allowed by the API and points there once its DNS is set.

4.7c Errors on this page

Code	Cause	Fix
`checkout_signature_invalid`	The link was changed after signing, its key was revoked, the publishable key is another account's, or `redirectUrl` is off its origins	Sign a new one
`checkout_expired`	Past `exp`	Sign a new one

Next: 9.2 Your account.

Last verified: 5 November 2026, against the SDK source and the API it runs.

5.1 Buy BTC from an EVM chain

After this page you can run a whole buy over plain HTTP, step by step, in curl, TypeScript or Python.

Before you start: an API key, a wallet on Base, and the 2.1 Quickstart. **Time:** 30 minutes.

The short path. With the TypeScript SDK the whole flow is one call. The SDK runs every check on this page before the wallet is asked to sign:

TypeScript

```
import { createMaxyClient, evmSignerFromWalletClient } from '@maxy-labs/sdk';

const maxy = createMaxyClient(); // reads MAXY_API_KEY
const evm = evmSignerFromWalletClient(walletClient); // the wallet client from the setup below,
or the user's wallet
const { order, status } = await maxy.swap.execute({ from: 'USDC on base', to: 'BTC', amount:
'10', evm, btcAddress });
```

`buyBtcWithEvm` from `@maxy-labs/sdk/evm` does the same over the HTTP API, with one progress event per step and a resume path (flow helpers). A firm quote also carries `signing`: what the wallet is asked for, in one shape on every route.

The rest of this page is the same flow with raw calls. USDC has its own page: gasless buy.

5.1a Setup

Every sample in this guide uses your key, a test wallet of yours (`TEST_WALLET_KEY`, a hex private key) and your Bitcoin address (`MY_BTC_ADDRESS`, `bc1q…`). With a test key (`mxk_test_…`) nothing real happens and the wallet can be empty (sandbox); never send a sandbox `transaction`. With a live key, use a test wallet holding a few dollars.

curl

```
export BASE=https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api
export MAXY_API_KEY=mxk_live_… TEST_WALLET_KEY=0x… MY_BTC_ADDRESS=bc1q…
export WALLET=$(cast wallet address --private-key "$TEST_WALLET_KEY")   # Foundry's cast signs
in the curl samples
```

TypeScript

```
import { createWalletClient, hashTypedData, http } from 'viem';
import { privateKeyToAccount } from 'viem/accounts';
import { base } from 'viem/chains';

const BASE = 'https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api';
const KEY = process.env.MAXY_API_KEY!;
const account = privateKeyToAccount(process.env.TEST_WALLET_KEY as `0x${string}`);
const walletClient = createWalletClient({ account, chain: base, transport: http() });
const btcAddress = process.env.MY_BTC_ADDRESS!;
```

Python

```
import os, requests
from eth_account import Account
from eth_account.messages import encode_typed_data

BASE = "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api"
KEY = os.environ["MAXY_API_KEY"]
account = Account.from_key(os.environ["TEST_WALLET_KEY"])
btc_address = os.environ["MY_BTC_ADDRESS"]
RPC = "https://mainnet.base.org"  # any Base RPC

def rpc(method, *params):
    return requests.post(RPC, json={"jsonrpc": "2.0", "id": 1, "method": method, "params":
list(params)}).json()["result"]

def send_transaction(tx):
    """Signs a transaction step with the test wallet and sends it; returns its hash."""
    req = {"from": account.address, "to": tx["to"], "data": tx["data"], "value":
hex(int(tx.get("value") or 0))}
    signed = account.sign_transaction({
        "to": tx["to"], "data": tx["data"], "value": int(tx.get("value") or 0), "chainId":
tx["chainId"],
        "nonce": int(rpc("eth_getTransactionCount", account.address, "pending"), 16),
        "gas": int(rpc("eth_estimateGas", req), 16) * 12 // 10, "gasPrice":
int(rpc("eth_gasPrice"), 16),
    })
    return rpc("eth_sendRawTransaction", "0x" + bytes(signed.raw_transaction).hex())
```

Two routes buy BTC from an EVM chain. The quote picks the route; your code follows its `steps`.

Paying with	Route	The wallet	Gas
ETH (Base, Arbitrum, Robinhood Chain), BNB, AVAX	`evm-btc-escrow`	signs the intent, then sends one escrow transaction	the user pays it

Paying with	Route	The wallet	Gas
USDC (Base, Arbitrum)	`evm-btc-gasless`	signs a USDC permit (when Permit2's allowance does not cover the order) and a Permit2 transfer	the solver pays it

---

5.1b List pairs

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/pairs"
```

TypeScript

```
const { pairs } = await fetch(`${BASE}/pairs`).then((r) => r.json());
```

Python

```
pairs = requests.get(f"{BASE}/pairs").json()["pairs"]
```

Response


```
{
  "pairs": [
    {
      "from": {
        "chain": 8453,
        "token": "0x0000000000000000000000000000000000000000",
        "symbol": "ETH",
        "decimals": 18,
        "chainName": "Base"
      },
      "to": {
        "chain": "bitcoin",
        "token": "BTC",
        "symbol": "BTC",
        "decimals": 8,
        "chainName": "Bitcoin"
      },
      "label": "ETH on Base → BTC"
    },
    {
      "from": {
        "chain": 8453,
        "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
        "symbol": "USDC",
        "decimals": 6,
        "chainName": "Base"
      },
      "to": {
        "chain": "bitcoin",
        "token": "BTC",
        "symbol": "BTC",
        "decimals": 8,
        "chainName": "Bitcoin"
      },
      "label": "USDC on Base → BTC"
    },
    {
      "from": {
        "chain": 42161,
        "token": "0x0000000000000000000000000000000000000000",
        "symbol": "ETH",
        "decimals": 18,
        "chainName": "Arbitrum"
      },
      "to": {
        "chain": "bitcoin",
        "token": "BTC",
```

```
        "symbol": "BTC",
        "decimals": 8,
        "chainName": "Bitcoin"
      },
      "label": "ETH on Arbitrum → BTC"
    },
    {
      "from": {
        "chain": 42161,
        "token": "0xaf88d065e77c8cC2239327C5EDb3A432268e5831",
        "symbol": "USDC",
        "decimals": 6,
        "chainName": "Arbitrum"
      },
      "to": {
        "chain": "bitcoin",
        "token": "BTC",
        "symbol": "BTC",
        "decimals": 8,
        "chainName": "Bitcoin"
      },
      "label": "USDC on Arbitrum → BTC"
    },
    "…"
  ]
}
```


`GET /pairs`: a real answer from the live API, 2 November 2026 (`x-request-id` 95f9a6a9-4b2e-433c-91b5-5ca34ab1e6e8); the first four pairs.

Cache the list for up to 5 minutes: the API sends `max-age=300`. The chains, tokens and contracts behind each pair are on chains and tokens and contracts.

5.1c Show a price

While the user types, ask for a price. It takes your key and no wallet, carries no `quoteId`, and reserves nothing.

curl

```
curl -s "$BASE/price?from=USDC%20on%20base&to=BTC&amount=10"
```

TypeScript

```
const price = await fetch(`${BASE}/price?${new URLSearchParams({ from: 'USDC on base', to:
'BTC', amount: '10' })}`).then((r) => r.json());
// price.formatted.receive -> '0.0001244 BTC'
```

Python

```
price = requests.get(f"{BASE}/price", params={"from": "USDC on base", "to": "BTC", "amount":
"10"}).json()
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": {
    "chain": 8453,
    "chainName": "Base",
    "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
    "symbol": "USDC",
    "decimals": 6,
    "caip2": "eip155:8453"
  },
  "to": {
    "chain": "bitcoin",
    "chainName": "Bitcoin",
    "token": "BTC",
    "symbol": "BTC",
    "decimals": 8,
    "caip2": "bip122:000000000019d6689c085ae165831e93"
  },
  "fromAmount": "10000000",
  "toAmount": "12468",
  "toAmountMin": "12468",
  "formatted": {
    "send": "10.00 USDC on Base",
    "receive": "0.00012468 BTC",
    "pair": "USDC on Base → BTC"
  },
  "expiresAt": 1793636689,
  "limits": {
    "minSats": 546
  }
}
```


`GET /price` for 10 USDC on Base: a real answer from the live API, 2 November 2026 (`x-request-id` 27c7050b-23e7-478a-9eb5-3d9861cb4b95).

5.1d Get a quote

When the user moves to confirm, get the quote. It names the paying wallet and the receiving Bitcoin address, and needs your key.

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" \
  --data-urlencode "from=ETH on base" --data-urlencode "to=BTC" --data-urlencode "amount=0.004"
\
  --data-urlencode "fromAddress=$WALLET" --data-urlencode "toAddress=$MY_BTC_ADDRESS" >
quote.json
```

TypeScript

```
const q = new URLSearchParams({ from: 'ETH on base', to: 'BTC', amount: '0.004', fromAddress:
account.address, toAddress: btcAddress });
const quote = await fetch(`${BASE}/quote?${q}`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
quote = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, params={
    "from": "ETH on base", "to": "BTC", "amount": "0.004",
    "fromAddress": account.address, "toAddress": btc_address,
}).json()
```

Response


```
{
  "quoteId": "0xceef574866d0cb1c94385d67dd6c1b525288d3b7db33bcf85b752274499d6790",
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": {
    "chain": 8453,
    "chainName": "Base",
    "token": "0x0000000000000000000000000000000000000000",
    "symbol": "ETH",
    "decimals": 18,
    "caip2": "eip155:8453"
  },
  "to": {
    "chain": "bitcoin",
    "chainName": "Bitcoin",
    "token": "BTC",
    "symbol": "BTC",
    "decimals": 8,
    "caip2": "bip122:000000000019d6689c085ae165831e93"
  },
  "fromAmount": "4000000000000000",
  "toAmount": "11259",
  "toAmountMin": "11259",
  "formatted": {
    "send": "0.004 ETH on Base",
    "receive": "0.00011259 BTC",
    "pair": "ETH on Base → BTC"
  },
  "expiresAt": 1793636692,
  "limits": {
    "minSats": 546
  },
  "order": {
    "route": "evm-btc-escrow",
    "orderId": "0xceef574866d0cb1c94385d67dd6c1b525288d3b7db33bcf85b752274499d6790",
    "chainId": 8453,
    "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
    "intent": {
      "side": 0,
      "maker": "0x2A7A049261E1DB2C48885AbeF0e37b9873cb9A24",
      "asset": "0x0000000000000000000000000000000000000000",
      "scriptHash": "0x874e5c457b11d8bfba21fce0322061d016f20128abcb17bd41fa76e4665d96a3",
      "satAmount": "11259",
      "designatedFiller": "0x0000000000000000000000000000000000000000",
      "pricingMode": 0,
      "limitPrice": "355182427036",
      "startPrice": "0",
```

```
      "endPrice": "0",
      "decayStartTime": "0",
      "decayEndTime": "0",
      "slippageBps": 50,
      "oracleDiscountBps": 0,
      "bondBps": 100,
      "requiredConfirmations": 1,
      "settlementTimeoutBlocks": 144,
      "deadline": "1793643832",
      "nonce": "0",
      "oracleConfigHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
    },
    "status": "/status?
orderId=0xceef574866d0cb1c94385d67dd6c1b525288d3b7db33bcf85b752274499d6790&chainId=8453&route=evm-
btc-escrow"
  },
  "issues": {
    "balance": {
      "token": "0x0000000000000000000000000000000000000000",
      "actual": "0",
      "expected": "4018993940726289"
    },
    "allowance": null,
    "orderCap": null
  },
  "steps": [
    {
      "id": "intent",
      "kind": "signature",
      "description": "Sign the order (EIP-712). No gas.",
      "signatureKind": "eip712",
      "typedData": {
        "domain": {
          "name": "BitcoinIntentSettler",
          "version": "6",
          "chainId": 8453,
          "verifyingContract": "0xeC5a05b28dd848F628e19CB26Df32f053e64ab86"
        },
        "types": {
          "Intent": [
            {
              "name": "side",
              "type": "uint8"
            },
            {
              "name": "maker",
              "type": "address"
            },
            {
```

```
      "name": "asset",
      "type": "address"
    },
    {
      "name": "scriptHash",
      "type": "bytes32"
    },
    {
      "name": "satAmount",
      "type": "uint64"
    },
    {
      "name": "designatedFiller",
      "type": "address"
    },
    {
      "name": "pricingMode",
      "type": "uint8"
    },
    {
      "name": "limitPrice",
      "type": "uint256"
    },
    {
      "name": "startPrice",
      "type": "uint256"
    },
    {
      "name": "endPrice",
      "type": "uint256"
    },
    {
      "name": "decayStartTime",
      "type": "uint64"
    },
    {
      "name": "decayEndTime",
      "type": "uint64"
    },
    {
      "name": "slippageBps",
      "type": "uint16"
    },
    {
      "name": "oracleDiscountBps",
      "type": "uint16"
    },
    {
      "name": "bondBps",
```

```
              "type": "uint16"
            },
            {
              "name": "requiredConfirmations",
              "type": "uint16"
            },
            {
              "name": "settlementTimeoutBlocks",
              "type": "uint32"
            },
            {
              "name": "deadline",
              "type": "uint64"
            },
            {
              "name": "nonce",
              "type": "uint256"
            },
            {
              "name": "oracleConfigHash",
              "type": "bytes32"
            }
          ]
        },
        "primaryType": "Intent",
        "message": {
          "side": 0,
          "maker": "0x2A7A049261E1DB2C48885AbeF0e37b9873cb9A24",
          "asset": "0x00000000000000000000000000000000000000000000",
          "scriptHash": "0x874e5c457b11d8bfba21fce0322061d016f20128abcb17bd41fa76e4665d96a3",
          "satAmount": "11259",
          "designatedFiller": "0x0000000000000000000000000000000000000000000000",
          "pricingMode": 0,
          "limitPrice": "355182427036",
          "startPrice": "0",
          "endPrice": "0",
          "decayStartTime": "0",
          "decayEndTime": "0",
          "slippageBps": 50,
          "oracleDiscountBps": 0,
          "bondBps": 100,
          "requiredConfirmations": 1,
          "settlementTimeoutBlocks": 144,
          "deadline": "1793643832",
          "nonce": "0",
          "oracleConfigHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
        }
      },
```

```
      "hash": "0xceef574866d0cb1c94385d67dd6c1b525288d3b7db33bcf85b752274499d6790"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signature; you get back the transaction to send.",
      "method": "POST",
      "path": "/orders",
      "body": {
        "order": {
          "route": "evm-btc-escrow",
          "orderId": "0xceef574866d0cb1c94385d67dd6c1b525288d3b7db33bcf85b752274499d6790",
          "chainId": 8453,
          "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
          "intent": {
            "side": 0,
            "maker": "0x2A7A049261E1DB2C48885AbeF0e37b9873cb9A24",
            "asset": "0x0000000000000000000000000000000000000000",
            "scriptHash": "0x874e5c457b11d8bfba21fce0322061d016f20128abcb17bd41fa76e4665d96a3",
            "satAmount": "11259",
            "designatedFiller": "0x0000000000000000000000000000000000000000",
            "pricingMode": 0,
            "limitPrice": "355182427036",
            "startPrice": "0",
            "endPrice": "0",
            "decayStartTime": "0",
            "decayEndTime": "0",
            "slippageBps": 50,
            "oracleDiscountBps": 0,
            "bondBps": 100,
            "requiredConfirmations": 1,
            "settlementTimeoutBlocks": 144,
            "deadline": "1793643832",
            "nonce": "0",
            "oracleConfigHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
          },
          "status": "/status?
orderId=0xceef574866d0cb1c94385d67dd6c1b525288d3b7db33bcf85b752274499d6790&chainId=8453&route=evm-
btc-escrow"
        },
        "signatures": {
          "intent": "<the intent signature>"
        }
      }
    },
    {
      "id": "escrow",
      "kind": "transaction",
```

```
      "description": "Send the escrow transaction (4018993940726289 wei). Then POST
/orders/submit with its hash.",
      "chainId": 8453,
      "transaction": null
    }
  ]
}
```


`GET /quote` for 0.004 ETH on Base, from an empty test wallet (so `issues.balance` is set): a real answer from the live API, 2 November 2026 (`x-request-id` 802219f5-0857-4b74-a9d2-a3ad5b458a24).

Every field is on EVM buy: fields and failures.

Quoting USDC instead gives a `permit` and a `witness` signature step, and only for a wallet that holds the USDC: for an empty wallet, `quoteId`, `order` and `steps` are `null` and `issues.balance` says how much is missing.

5.1e Check the order size

If `issues.orderCap` is set, the order is above your key's cap. Show its `message` and stop:

Response


```
{ "capUsd": 100, "orderUsd": 149.62, "message": "Order size is capped for this API key. Try $100
or less." }
```


`issues.orderCap` of a real quote for $150 of USDC on Base with a $100-capped key.

Below the smallest order, the quote itself answers `422 amount_out_of_range`.

5.1f Allowance

Follow the `steps` array in order and you do not need to tell the cases apart. The cases.

5.1g Sign

Each `signature` step carries the EIP-712 `typedData` exactly as the wallet will show it, and its `hash`. Check the hash, then ask the wallet to sign (`eth_signTypedData_v4`):

curl

```
INTENT_SIG=$(cast wallet sign --private-key "$TEST_WALLET_KEY" --data "$(jq -c '.steps[] |
select(.id == "intent") | .typedData' quote.json)")
```

TypeScript

```
if (!quote.steps) throw new Error(quote.issues?.orderCap?.message ?? 'Not enough funds for this
order.');
const signatures: Record<string, string> = {};
for (const step of quote.steps.filter((s) => s.kind === 'signature')) {
  if (hashTypedData(step.typedData) !== step.hash) throw new Error(`step ${step.id} does not
hash to ${step.hash}`);
  signatures[step.id] = await walletClient.signTypedData(step.typedData);
}
```

Python

```
from eth_utils import keccak

if not quote["steps"]:
    raise SystemExit((quote["issues"].get("orderCap") or {}).get("message", "Not enough funds
for this order."))
signatures = {}
for step in (s for s in quote["steps"] if s["kind"] == "signature"):
    message = encode_typed_data(full_message=step["typedData"])
    digest = "0x" + keccak(b"\x19" + message.version + message.header + message.body).hex()
    assert digest == step["hash"].lower()  # check what will be signed before signing it
    signatures[step["id"]] = "0x" + bytes(account.sign_message(message).signature).hex()
```

Response


```
{
  "maker": "0x5A9e8EF14C1980F0997aEb8Ad0Fb8d31F3F09D42",
  "quoteId": "0x2e10bcc8c80999d7cc79d40aebc6c8a3657fe2b5c7cec72de93247c8bb685b5f",
  "expiresAt": 1793635183,
  "signed": [
    {
      "id": "intent",
      "hash": "0x2e10bcc8c80999d7cc79d40aebc6c8a3657fe2b5c7cec72de93247c8bb685b5f",
      "signature":
"0xa6240bd22b70a7ac28bcc0c340ed2022d8ff95c7ef7653af0c1bc6ce80f5577135c47233e16ad69b45a7bcc15b2f90b3
    }
  ],
  "signatures": {
    "intent":
"0xa6240bd22b70a7ac28bcc0c340ed2022d8ff95c7ef7653af0c1bc6ce80f5577135c47233e16ad69b45a7bcc15b2f90b3
  }
}
```


What the kit's `04-sign.ts` printed on 2 November 2026: signed locally by a throwaway test wallet; nothing was sent to Maxy.

What each signature binds is on security model: the paying wallet, the coin, the sats, the price and its band. It also binds the solver's bond, the Bitcoin script that must receive the BTC, and the deadline. The words to show are on UX guidelines.

5.1h Create the order

Send the quote's `order` unchanged, with the signatures keyed by step id:

curl

```
curl -s -X POST "$BASE/orders" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d "$(jq -c --arg sig "$INTENT_SIG" '{order: .order, signatures: {intent: $sig}}' quote.json)"
> placed.json
```

TypeScript

```
const placed = await fetch(`${BASE}/orders`, {
  method: 'POST',
  headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' },
  body: JSON.stringify({ order: quote.order, signatures }),
}).then((r) => r.json());
```

Python

```
placed = requests.post(f"{BASE}/orders", headers={"x-maxy-key": KEY},
                       json={"order": quote["order"], "signatures": signatures}).json()
```

Response


```
{
  "orderId": "0xb598c04cef806eab70be51b4b535b19863ad0521558b518bb57e0beb782e7d9f",
  "route": "evm-btc-escrow",
  "expiresAt": 1793643833,
  "transaction": {
    "chainId": 8453,
    "to": "0xeC5a05b28dd848F628e19CB26Df32f053e64ab86",
    "data":
"0x2abe5f670000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    "value": "4018993940726289",
    "from": "0xAF7F918699c851b8E3dBA3FD7c0132f6fD523C0A"
  },
  "simulation": {
    "ok": false,
    "error": "The wallet does not hold enough of the coin for this deposit plus gas."
  },
  "next": {
    "method": "POST",
    "path": "/orders/submit",
    "body": {
      "order": {
        "route": "evm-btc-escrow",
        "orderId": "0xb598c04cef806eab70be51b4b535b19863ad0521558b518bb57e0beb782e7d9f",
        "chainId": 8453,
        "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
        "intent": {
          "side": 0,
          "maker": "0xAF7F918699c851b8E3dBA3FD7c0132f6fD523C0A",
          "asset": "0x0000000000000000000000000000000000000000",
          "scriptHash": "0x874e5c457b11d8bfba21fce0322061d016f20128abcb17bd41fa76e4665d96a3",
          "satAmount": "11259",
          "designatedFiller": "0x0000000000000000000000000000000000000000",
          "pricingMode": 0,
          "limitPrice": "355182427036",
          "startPrice": "0",
          "endPrice": "0",
          "decayStartTime": "0",
          "decayEndTime": "0",
          "slippageBps": 50,
          "oracleDiscountBps": 0,
          "bondBps": 100,
          "requiredConfirmations": 1,
          "settlementTimeoutBlocks": 144,
          "deadline": "1793643833",
          "nonce": "0",
          "oracleConfigHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
```

```
        },
        "status": "/status?
orderId=0xb598c04cef806eab70be51b4b535b19863ad0521558b518bb57e0beb782e7d9f&chainId=8453&route=evm-
btc-escrow"
      },
      "signatures": {
        "intent":
"0x9f46794543f8974879690fa9963173f0baa1c17ec52b84decc87b4ffe4af64894d02bb263f8cbe0a05b4a2f47668c71d
      },
      "txHash": "<the hash of the transaction you sent>"
    }
  }
}
```


`POST /orders` for an order of 0.004 ETH on Base (a separate run of the same steps), signed by a throwaway test wallet that holds nothing, so the simulation says so and nothing was sent: a real answer from the live API, 2 November 2026 (`x-request-id` 9fd0444c-af81-4881-8b06-c08c7aee9c79).

The same call in the sandbox, where the simulation passes and the transaction is marked `simulated`:

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "expiresAt": 1794542691,
  "transaction": {
    "chainId": 8453,
    "from": "0xdd0d268Abc55f328F88f1Ee2e2f0850bFf43686b",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": {
    "ok": true
  },
  "next": {
    "method": "POST",
    "path": "/orders/submit",
    "body": {
      "order": "…",
      "signatures": "…",
      "txHash": "<transaction.simulatedHash>"
    }
  },
  "expiresAtIso": "2026-11-13T04:04:51.000Z",
  "refundAt": 1794542691,
  "sandbox": true
}
```


`POST /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` e6836a36-5e96-48f0-a94d-36486fe8db98), with long fields folded.

- **USDC (gasless):** the answer is `{ orderId, submitted: true, status }`. The solver opens the escrow and pays the gas; go to 5.1j (gasless buy).
- **ETH:** the answer carries the escrow `transaction` for the user's wallet to send, and its `simulation` from that wallet. If `simulation.ok` is false, do not send it; its `error` says why (usually the balance).
- **Retries are safe.** The same signed order returns the same `orderId`; the order id is the idempotency key.

- **Timing:** post within the quote's 60 seconds. After them the answer is `409 quote_expired`: get a new quote (5.1d) and sign again. The order's own deadline is 2 hours after signing (returned as `expiresAt`), and a solver fills it only at the price the user signed.

5.1i Send the escrow (ETH only)

The user's wallet sends `placed.transaction`. Then tell Maxy its hash, so the solver starts at once (it also watches the chain):

TypeScript

```
const { from: _from, ...tx } = placed.transaction; // the wallet client sends from its own
account
const txHash = await walletClient.sendTransaction({ ...tx, value: BigInt(tx.value) });
await fetch(`${BASE}/orders/submit`, {
  method: 'POST',
  headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' },
  body: JSON.stringify({ ...placed.next.body, txHash }),
});
```

Python

```
tx_hash = send_transaction(placed["transaction"])  # defined in the setup above; or your own key
management
requests.post(f"{BASE}/orders/submit", headers={"x-maxy-key": KEY}, json={**placed["next"]
["body"], "txHash": tx_hash})
```

curl

```
curl -s -X POST "$BASE/orders/submit" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d "$(jq -c --arg sig "$INTENT_SIG" --arg h "$TX_HASH" '{order: .order, signatures: {intent:
$sig}, txHash: $h}' quote.json)"
```

Response

```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "notified": true,
  "status": "/status?
orderId=0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3&chainId=8453&route=evm-
btc-escrow",
  "sandbox": true
}
```

`POST /orders/submit`: a real sandbox answer, 13 November 2026 (`x-request-id` 2e7f1e4c-d9f9-4350-9ad5-13c93f7acf8a).

5.1j Watch the order

Webhooks deliver each change as `order.updated` (track orders and receive webhooks). While the user watches, poll every 2 seconds:

curl

```
ORDER_ID=$(jq -r .orderId placed.json)
curl -s "$BASE/status?orderId=$ORDER_ID&chainId=8453"
```

TypeScript

```
const status = await fetch(`${BASE}/status?${new URLSearchParams({ orderId: placed.orderId,
chainId: '8453' })}`).then((r) => r.json());
```

Python

```
status = requests.get(f"{BASE}/status", params={"orderId": placed["orderId"], "chainId":
8453}).json()
```

`state`	Typical time	Webhook
`pending`	one block on the paying chain (about 2 seconds on Base), then the solver commits	`order.updated`
`in_progress`	seconds: the solver is paying	`order.updated`
`completed`	as soon as the BTC is sent: `btcTxId` is set	`order.updated`

All nine states, with who acts next, are on order lifecycle. Mid-flight, from the sandbox:

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "state": "in_progress",
  "rawStatus": "SANDBOX_FILLING",
  "isComplete": false,
  "btcTxId": null,
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "refundAt": null,
  "refundAtIso": null,
  "refundAfterBlock": null,
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 798bcbbe-a611-460e-9ace-d71b44de4264).

5.1k Completed

Response


```
{
  "orderId": "0xed3985cf8b6dcc9a30448c9faaefbaf3a2f2ef9c06f50ce1f46dd955212ce2ea",
  "route": "evm-btc-gasless",
  "state": "completed",
  "rawStatus": "SETTLED",
  "isComplete": true,
  "btcTxId": "6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea3f",
  "txs": [
    {
      "label": "escrow",
      "hash": "0xf103819940fef19e88f9c0eafb24a52bad8de86c277e18e8e2e60a8594c7f8d1"
    },
    {
      "label": "btc_payout",
      "hash": "6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea3f"
    },
    {
      "label": "settlement",
      "hash": "0xef66f624a34cff33a5d5660e1666e15669bc02dfc8b0a27fa827e8b268e702f2"
    }
  ]
}
```


`GET /status` for a real completed order: $10 of USDC on Base to 10,387 sats: a real answer from the live API, 2 November 2026 (`x-request-id` eb15599f-74c7-4c77-95e2-8f50240f3b0a).

- `btcTxId` is the Bitcoin payment to the user: link it as `https://mempool.space/tx/<btcTxId>`.
- A receipt shows: what was paid (`fromAmount`, chain), what arrived (`toAmount` in BTC), the receiving address, the `btcTxId`, the escrow transaction, and the `orderId`.
- The order is `completed` the moment `btcTxId` is set. The user's wallet shows the BTC after its first confirmation, about 10 minutes. The settlement that follows (`rawStatus` `BTC_SENT`, then `SETTLED`) needs nothing from the user.

5.1l Errors on this page

Code	HTTP	Step	Cause	Fix
`invalid_input`	400	3, 7, 8	Missing addresses; a malformed order, signature or `txHash`	Fix the input; send `order` unchanged
`auth_required`	401	3, 7, 8	No key	Send `x-maxy-key`

Code	HTTP	Step	Cause	Fix
`amount_out_of_range`	422	2, 3	Below the smallest order	Raise the amount
`order_cap_exceeded`	422	7	Above your key's cap	Order less
`quote_expired`	409	7	The quote is older than 60 seconds	New quote
`wallet_mismatch`	400	7	A signature that does not recover to `fromAddress`	Sign with the paying wallet
`rate_limited`	429	any	Over your rate limit	Wait `Retry-After` seconds

Next: 5.2 EVM buy: fields and failures.

Last verified: 13 November 2026, against the sandbox.

5.2 EVM buy: fields and failures

After this page you can read every field of a quote, tell the allowance cases apart, and handle each way a buy can fail.

Before you start: 5.1 Buy BTC from an EVM chain. **Time:** 5 minutes.

5.2a Quote fields

Field	Meaning
`quoteId`	The order id this quote becomes (the intent hash); `null` when nothing can be ordered
`expiresAt`	Unix seconds; 60 seconds after the quote
`fromAmount`, `toAmount`	What the user pays and what arrives, in the smallest unit. `toAmount` is executable
`toAmountMin`	Equal to `toAmount`
`formatted`	The amounts in words, for display
`limits.minSats`	The smallest order
`issues.balance`	Set when the wallet holds less than `fromAmount` (`actual`, `expected`)
`issues.allowance`	Set when the quote starts with an `approve` transaction: only for a token without an EIP-2612 permit (USDC on Base and Arbitrum has one)
`issues.orderCap`	Set when the order is above your key's order size cap; `steps` is then `null`
`order`	The order to send back unchanged to `POST /orders`
`steps`	What the wallet does, in order: `signature` (EIP-712 `typedData` and its `hash`), `transaction`, `post`

5.2b Allowance cases

- **ETH, BNB, AVAX** need none: the escrow transaction carries the value.
- **USDC** moves through Permit2. When Permit2's USDC allowance does not cover the order, the wallet first signs an EIP-2612 permit (`id: "permit"`) for exactly this order's amount. It then always signs the Permit2 transfer (`id: "witness"`), bound to this order.
- A token without a permit gets an `approve` transaction (`id: "approve"`) for this order's amount instead, and `issues.allowance` is set: send it and wait for it to confirm. USDC on Base and

Arbitrum never needs one.

5.2c Failure branches

Quote expired. Check `expiresAt` before signing and get a new quote when it has passed ("Prices moved. Here is a new quote."). Nothing is sent.

Response


```
{
  "error": {
    "code": "quote_expired",
    "message": "This quote expired (a quote lasts 60 seconds). Nothing was sent. Get a new quote
and sign that.",
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#quote_expired",
    "details": {
      "freshQuote": {
        "route": "evm-btc-escrow",
        "liquidityAvailable": true,
        "from": {
          "chain": 8453,
          "chainName": "Base",
          "token": "0x0000000000000000000000000000000000000000",
          "symbol": "ETH",
          "decimals": 18,
          "caip2": "eip155:8453",
          "assetId": "eip155:8453/slip44:60"
        },
        "to": {
          "chain": "bitcoin",
          "chainName": "Bitcoin",
          "token": "BTC",
          "symbol": "BTC",
          "decimals": 8,
          "caip2": "bip122:000000000019d6689c085ae165831e93",
          "assetId": "bip122:000000000019d6689c085ae165831e93/slip44:0"
        },
        "fromAmount": "4000000000000000",
        "toAmount": "10882",
        "toAmountMin": "10882",
        "formatted": {
          "send": "0.004 ETH on Base",
          "receive": "0.00010882 BTC",
          "pair": "ETH on Base → BTC"
        },
        "expiresAt": 1794535605,
        "expiresAtIso": "2026-11-13T02:06:45.000Z",
        "limits": {
          "minSats": 546
        },
        "quoteId": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
        "order": {
          "route": "evm-btc-escrow",
          "orderId": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
          "chainId": 8453,
```

```
          "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
          "intent": {
            "side": 0,
            "maker": "0xC10B40524740Bc968e67D90c2088D5EF30635f2e",
            "asset": "0x0000000000000000000000000000000000000000",
            "scriptHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
            "satAmount": "10882",
            "designatedFiller": "0x0000000000000000000000000000000000000000",
            "pricingMode": 0,
            "limitPrice": "0",
            "startPrice": "0",
            "endPrice": "0",
            "decayStartTime": "0",
            "decayEndTime": "0",
            "slippageBps": 0,
            "oracleDiscountBps": 0,
            "bondBps": 0,
            "requiredConfirmations": 1,
            "settlementTimeoutBlocks": 0,
            "deadline": "1794542745",
            "nonce": "0",
            "oracleConfigHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
          },
          "status": "/status?
orderId=0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff&chainId=8453&route=evm-
btc-escrow",
          "sandbox": {
            "scenario": "success",
            "fromAddress": "0xC10B40524740Bc968e67D90c2088D5EF30635f2e",
            "toAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
            "fromToken": "0x0000000000000000000000000000000000000000",
            "toToken": "BTC",
            "fromAmount": "4000000000000000",
            "toAmount": "10882",
            "usd": 7.7
          },
          "quote": {
            "expiresAt": 1794535605,
            "id": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
            "token": "52f407983e777c911ca63f707a945fce1c42851629c7e12e72d4eb3fa9e2f0ec",
            "ask":
"from=ETH+on+base&to=BTC&amount=0.004&fromAddress=0xC10B40524740Bc968e67D90c2088D5EF30635f2e&toAddr
          }
        },
        "issues": {
          "balance": null,
          "allowance": null,
          "orderCap": null
```

```
        },
        "steps": [
          {
            "id": "intent",
            "kind": "signature",
            "description": "Sign the order (EIP-712). No gas. Sandbox: this signature authorizes
nothing.",
            "signatureKind": "eip712",
            "typedData": {
              "domain": {
                "name": "Maxy Sandbox",
                "version": "1",
                "chainId": 8453,
                "verifyingContract": "0x0000000000000000000000000000000000000000"
              },
              "types": {
                "MaxySandboxOrder": [
                  {
                    "name": "orderId",
                    "type": "string"
                  },
                  {
                    "name": "step",
                    "type": "string"
                  },
                  {
                    "name": "notice",
                    "type": "string"
                  }
                ]
              },
              "primaryType": "MaxySandboxOrder",
              "message": {
                "orderId": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
                "step": "intent",
                "notice": "Maxy sandbox. This moves nothing and cannot be spent."
              }
            },
            "hash": "0xdab6834d468daab77a35596f3f7dabd8a554e78a788ca0fbf5aa40d65c331a61"
          },
          {
            "id": "order",
            "kind": "post",
            "description": "Send the order and its signature; you get back the transaction to
send.",
            "method": "POST",
            "path": "/orders",
            "body": {
              "order": {
```

```
                "route": "evm-btc-escrow",
                "orderId": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
                "chainId": 8453,
                "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
                "intent": {
                  "side": 0,
                  "maker": "0xC10B40524740Bc968e67D90c2088D5EF30635f2e",
                  "asset": "0x0000000000000000000000000000000000000000",
                  "scriptHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
                  "satAmount": "10882",
                  "designatedFiller": "0x0000000000000000000000000000000000000000",
                  "pricingMode": 0,
                  "limitPrice": "0",
                  "startPrice": "0",
                  "endPrice": "0",
                  "decayStartTime": "0",
                  "decayEndTime": "0",
                  "slippageBps": 0,
                  "oracleDiscountBps": 0,
                  "bondBps": 0,
                  "requiredConfirmations": 1,
                  "settlementTimeoutBlocks": 0,
                  "deadline": "1794542745",
                  "nonce": "0",
                  "oracleConfigHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
                },
                "status": "/status?
orderId=0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff&chainId=8453&route=evm-
btc-escrow",
                "sandbox": {
                  "scenario": "success",
                  "fromAddress": "0xC10B40524740Bc968e67D90c2088D5EF30635f2e",
                  "toAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
                  "fromToken": "0x0000000000000000000000000000000000000000",
                  "toToken": "BTC",
                  "fromAmount": "4000000000000000",
                  "toAmount": "10882",
                  "usd": 7.7
                },
                "quote": {
                  "expiresAt": 1794535605,
                  "id": "0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
                  "token": "52f407983e777c911ca63f707a945fce1c42851629c7e12e72d4eb3fa9e2f0ec",
                  "ask":
"from=ETH+on+base&to=BTC&amount=0.004&fromAddress=0xC10B40524740Bc968e67D90c2088D5EF30635f2e&toAddr
                }
              },
```

```
              "signatures": {
                "intent": "<the intent signature>"
              }
            }
          },
          {
            "id": "escrow",
            "kind": "transaction",
            "description": "Sandbox: do not send the transaction. POST /orders/submit with its
simulatedHash.",
            "chainId": 8453,
            "transaction": null
          }
        ],
        "signing": {
          "kind": "eip712",
          "payloads": [
            {
              "id": "intent",
              "typedData": {
                "domain": {
                  "name": "Maxy Sandbox",
                  "version": "1",
                  "chainId": 8453,
                  "verifyingContract": "0x0000000000000000000000000000000000000000"
                },
                "types": {
                  "MaxySandboxOrder": [
                    {
                      "name": "orderId",
                      "type": "string"
                    },
                    {
                      "name": "step",
                      "type": "string"
                    },
                    {
                      "name": "notice",
                      "type": "string"
                    }
                  ]
                },
                "primaryType": "MaxySandboxOrder",
                "message": {
                  "orderId":
"0x53414e44424f58abb3c81cfa71434a5ba2e844f53b2270881735feb6b90214ff",
                  "step": "intent",
                  "notice": "Maxy sandbox. This moves nothing and cannot be spent."
                }
```

```
              },
              "hash": "0xdab6834d468daab77a35596f3f7dabd8a554e78a788ca0fbf5aa40d65c331a61"
            }
          ],
          "approval": null,
          "after": {
            "method": "POST",
            "path": "/orders"
          }
        },
        "sandbox": true
      }
    }
  },
  "requestId": "e7b47aa9-fcbe-4b03-9181-2e3717bd542b",
  "sandbox": true
}
```


`POST /orders` (HTTP 409): a real sandbox answer, 13 November 2026 (`x-request-id` e7b47aa9-fcbe-4b03-9181-2e3717bd542b).

The user rejected the signature. The wallet throws (`4001` from an EIP-1193 provider; the SDK's `user_rejected`). Nothing was signed; return to the confirm screen. If the quote is still valid, the user can try again.

A signature from another wallet.

Response


```
{
  "error": {
    "code": "wallet_mismatch",
    "message": "The intent signature did not come from the paying wallet
(0xEaDd0b32dB895bb082739E7DCad7cB1ad4cbe391).",
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#wallet_mismatch"
  },
  "requestId": "f3aa0fd7-a2b9-40be-8ae1-a8df9ecba5b4",
  "sandbox": true
}
```


`POST /orders` (HTTP 400): a real sandbox answer, 13 November 2026 (`x-request-id` f3aa0fd7-a2b9-40be-8ae1-a8df9ecba5b4).

The solver did not fill. The order reads `refundable` (`rawStatus` `EXPIRED`, `FAILED` or `COMMIT_FAILED`): no solver committed before the escrow deadline (2 hours), or one committed and did not pay before the Bitcoin deadline (144 blocks after the commit). The user's wallet takes the coins

back from the settler (`swap.refund` in the SDK). An EVM buy never reads `failed` or `expired`. See refunds and failures.

Response


```
{
  "orderId": "0x53414e44424f58ec837ff3f2e3b5ff0db94324f8ff30f5e2d799de1f997273cd",
  "route": "evm-btc-escrow",
  "state": "refundable",
  "rawStatus": "SANDBOX_REFUNDABLE",
  "isComplete": false,
  "btcTxId": null,
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f58a6b9ebf67d835aeb77a71569a58bbd2b187dab0cf0ec272259"
    }
  ],
  "failureReason": {
    "code": "no_solver_commit",
    "message": "No solver committed before the escrow deadline. The escrow goes back to the
user."
  },
  "sandbox": true,
  "scenario": "refund_available",
  "refundAt": 1794535573,
  "refundAtIso": "2026-11-13T02:06:13.000Z",
  "refundAfterBlock": null,
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 43a5095c-8188-4b3e-9950-c54f470f71af).

5.2d Errors on this page

Code	HTTP	Cause	Fix
`quote_expired`	409	The quote is older than 60 seconds	New quote
`wallet_mismatch`	400	A signature that does not recover to `fromAddress`	Sign with the paying wallet
`user_rejected`		The wallet declined (`4001`)	Back to the confirm screen

Next: 5.3 Gasless buy.

Last verified: 13 November 2026, against the sandbox.

5.3 Gasless buy

After this page your user can buy BTC with USDC on Base or Arbitrum, signing only: no transaction, no gas.

Before you start: 5.1 Buy BTC from an EVM chain. **Time:** 10 minutes.

5.3a What changes

The flow is the EVM buy with the transaction taken out. A quote for USDC on Base or Arbitrum has `route` `evm-btc-gasless`.

	Escrow route (`evm-btc-escrow`)	**Gasless route (`evm-btc-gasless`)**
Paying with	ETH, BNB, AVAX	USDC on Base and Arbitrum
The wallet signs	The intent	A USDC permit (when Permit2's allowance is short), and a Permit2 transfer bound to this order
The wallet sends	One escrow transaction	Nothing
Who pays gas	The user	The solver
`POST /orders` answers	`transaction`, `simulation`, `next`	`submitted: true`
On screen	"Sign your order, then confirm one transaction."	"Sign in your wallet (once or twice). No gas."
After that	The user sends, you call `POST /orders/submit`	Nothing: the solver opens the escrow

Everything after the escrow opens is the same: the same states, the same webhook, the same refund rule.

5.3b Quote

A quote names its steps only for a wallet that holds the USDC. For an empty wallet, `quoteId`, `order` and `steps` are `null`, and `issues.balance` says what is missing. The sandbox reads no balances, so any wallet works there.

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" \
  --data-urlencode "from=USDC on base" -d to=BTC -d amount=10 \
  -d fromAddress="$WALLET" -d toAddress="$MY_BTC_ADDRESS" > quote.json
```

TypeScript

```
const q = new URLSearchParams({ from: 'USDC on base', to: 'BTC', amount: '10', fromAddress:
account.address, toAddress: btcAddress });
const quote = await fetch(`${BASE}/quote?${q}`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
quote = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, params={
    "from": "USDC on base", "to": "BTC", "amount": "10",
    "fromAddress": account.address, "toAddress": btc_address,
}).json()
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "10000000",
  "toAmount": "13669",
  "toAmountMin": "13669",
  "formatted": {
    "send": "10.00 USDC on Base",
    "receive": "0.00013669 BTC",
    "pair": "USDC on Base → BTC"
  },
  "expiresAt": 1794535579,
  "expiresAtIso": "2026-11-13T02:06:19.000Z",
  "limits": {
    "minSats": 546
  },
  "quoteId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
  "order": {
    "route": "evm-btc-gasless",
    "orderId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
    "chainId": 8453,
    "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
    "intent": "…",
    "status": "/status?
orderId=0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea&chainId=8453&route=evm-
btc-gasless",
    "sandbox": "…",
    "permit2Nonce": "0",
    "permit": null,
    "quote": {
      "expiresAt": 1794535579,
      "id": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
      "token": "566dc5457a68af703c8b023c4df61dc30a3083e35565f06c1e7bd59f2edd9fb7",
      "ask":
"from=USDC+on+base&to=BTC&amount=10&fromAddress=0xFc4Aa4354DE15df605F37395e4c3217Ff248077f&toAddres
    }
  },
  "issues": {
    "balance": null,
    "allowance": null,
    "orderCap": null
  },
  "steps": [
    {
      "id": "witness",
```

```
      "kind": "signature",
      "description": "Sign the transfer bound to this order. No gas. Sandbox: this signature
authorizes nothing.",
      "signatureKind": "eip712",
      "typedData": {
        "domain": {
          "name": "Maxy Sandbox",
          "version": "1",
          "chainId": 8453,
          "verifyingContract": "0x0000000000000000000000000000000000000000"
        },
        "types": "…",
        "primaryType": "MaxySandboxOrder",
        "message": {
          "orderId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
          "step": "witness",
          "notice": "Maxy sandbox. This moves nothing and cannot be spent."
        }
      },
      "hash": "0x1c3aa0811071c34e2f1b536f26ef9ed205ec6eda8598ecaa995d315f8629c707"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signatures. The solver opens the escrow and pays
the gas.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    }
  ],
  "signing": "…",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 686e7eae-9a18-4bba-a051-21694b9fd7d4), with long fields folded. In the sandbox the one signature is a `MaxySandboxOrder`, which authorizes nothing. Live, the steps are `permit` (when needed) and `witness`.

5.3c Sign

Sign every step of kind `signature`, keyed by its id. The code is the same as the EVM buy's signing section.

curl

```
SIG=$(cast wallet sign --private-key "$TEST_WALLET_KEY" --data "$(jq -c '.steps[] | select(.id
== "witness") | .typedData' quote.json)")
```

TypeScript

```
const signatures: Record<string, string> = {};
for (const step of quote.steps.filter((s) => s.kind === 'signature')) {
  if (hashTypedData(step.typedData) !== step.hash) throw new Error(`step ${step.id} does not
hash to ${step.hash}`);
  signatures[step.id] = await walletClient.signTypedData(step.typedData);
}
```

Python

```
signatures = {}
for step in (s for s in quote["steps"] if s["kind"] == "signature"):
    message = encode_typed_data(full_message=step["typedData"])
    signatures[step["id"]] = "0x" + bytes(account.sign_message(message).signature).hex()
```

5.3d Create the order

The signatures go to `POST /orders`. There is no second call. Retrying with the same signed order is safe: the solver opens one escrow per intent.

curl

```
curl -s -X POST "$BASE/orders" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d "$(jq -c --arg sig "$SIG" '{order: .order, signatures: {witness: $sig}}' quote.json)"
```

TypeScript

```
const placed = await fetch(`${BASE}/orders`, {
  method: 'POST',
  headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' },
  body: JSON.stringify({ order: quote.order, signatures }),
}).then((r) => r.json());
```

Python

```
placed = requests.post(f"{BASE}/orders", headers={"x-maxy-key": KEY},
                       json={"order": quote["order"], "signatures": signatures}).json()
```

Response


```
{
  "orderId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
  "route": "evm-btc-gasless",
  "submitted": true,
  "expiresAt": 1794542720,
  "status": "/status?
orderId=0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea&chainId=8453&route=evm-
btc-gasless",
  "expiresAtIso": "2026-11-13T04:05:20.000Z",
  "refundAt": 1794542720,
  "sandbox": true
}
```


`POST /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` 5c6402da-8cd6-4fc7-b170-d3a79a2b63fe).

`POST /orders/{id}/signature` is the same call with the order in the path: `{ order, signatures }`, or `{ order, signature }` on a route with one. The API keeps no state between a quote and its order, so the body carries the order.

5.3e Watch

The order walks `pending`, `in_progress`, `completed`, as on the EVM buy:

Response


```
{
  "orderId": "0x53414e44424f580368de4a878f43de42afaab9be7eb7b000fa516533d9ac1eea",
  "route": "evm-btc-gasless",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f58506805d05811929e2096c99c923fa8abbd3e1eb9e1a3083a83",
  "txs": [
    {
      "label": "btc_payout",
      "hash": "53414e44424f58506805d05811929e2096c99c923fa8abbd3e1eb9e1a3083a83"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 7bf23732-7cb5-43ed-86b4-10228ea3e1ba).

A real $10 gasless order, with its escrow, payout and settlement hashes, is on order lifecycle.

5.3f If the solver never opens the escrow

The user's USDC never left the wallet. The Permit2 signature lapses at the intent's deadline, 2 hours after signing. The order reads `refundable`, and there is nothing to refund (refunds and failures).

5.3g Errors on this page

Code	HTTP	Cause	Fix
`wallet_mismatch`	400	A signature that does not recover to `fromAddress`	Sign with the paying wallet
`quote_expired`	409	Posted after the quote's 60 seconds	New quote, sign again
`order_cap_exceeded`	422	Above your key's cap	Order less
`insufficient_funds`	422	The wallet holds less USDC than the order	Fund the wallet
`rate_limited`	429	Over your rate limit	Wait `Retry-After` seconds

Next: 5.4 Buy BTC from Solana.

Last verified: 13 November 2026, against the sandbox.

5.4 Buy BTC from Solana

After this page your user can buy BTC with SOL from their Solana wallet. The SOL stays locked in the Solana settlement program until the BTC payment is proven.

Before you start: an API key, a Solana wallet adapter in your app, and the 2.1 Quickstart. **Time:** 15 minutes.

The SDK path is first, then the same flow over plain HTTP. The steps match buy BTC from an EVM chain, except that the escrow is a Solana transaction.

5.4a The flow

1. **Price and quote.** Unchanged: `GET /price`, then `swap.quote({ from: 'SOL', to: 'BTC', amount })`.
2. **Sign.** The Solana wallet signs a readable quote request; the solver then opens its offer on chain.
3. **Escrow.** The wallet signs the escrow deposit: the SOL goes into the settlement program's escrow.
4. **Pay.** The solver pays the BTC to the user's address.
5. **Release.** The BTC payment is proven on Solana after one Bitcoin confirmation, and the program releases the SOL to the solver.

TypeScript

```
import { createMaxyClient } from '@maxy-labs/sdk';

const maxy = createMaxyClient({ apiKey: process.env.MAXY_API_KEY });
const q = await maxy.swap.quote({ from: 'SOL', to: 'BTC', amount: '1' });          // whole SOL,
or a bigint in lamports
const order = await maxy.swap.createOrder(q, { solana: solanaWallet, btcAddress: 'bc1q…' });
const done = await maxy.swap.waitForCompletion(order);   // rawStatus OPEN → PAYING → LOCKED →
PAID; state pending → in_progress → completed at PAID, when the BTC is sent
console.log(done.btcTxId);
```

`solanaWallet` is a signer built from your wallet adapter (signers). Native SegWit (`bc1q…`) and Taproot (`bc1p…`) addresses both receive on this route.

5.4b The same flow over HTTP

The samples use `BASE` and `KEY` from the setup. The signatures come from the user's Solana wallet, through your wallet adapter; a server signer works the same way. `steps` says what is next.

1. A price. With your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/price?from=SOL&to=BTC&amount=1"
```

TypeScript

```
const price = await fetch(`${BASE}/price?from=SOL&to=BTC&amount=1`).then((r) => r.json());
```

Python

```
price = requests.get(f"{BASE}/price", params={"from": "SOL", "to": "BTC", "amount": "1"}).json()
```

Response


```
{
  "route": "sol-btc",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "1000000000",
  "toAmount": "113170",
  "toAmountMin": "113170",
  "formatted": {
    "send": "1 SOL",
    "receive": "0.0011317 BTC",
    "pair": "SOL → BTC"
  },
  "expiresAt": 1794535533,
  "expiresAtIso": "2026-11-13T02:05:33.000Z",
  "sandbox": true
}
```


`GET /price`: a real sandbox answer, 13 November 2026 (`x-request-id` ca615f8b-e5aa-4beb-84ee-b6c426e82a1c), with long fields folded.

2. A quote. It names the Solana wallet and the Bitcoin address.

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" -d from=SOL -d to=BTC -d amount=1 \
  -d fromAddress="$SOL_WALLET" -d toAddress="$MY_BTC_ADDRESS" > quote.json
```

TypeScript

```
const q = new URLSearchParams({ from: 'SOL', to: 'BTC', amount: '1', fromAddress:
wallet.publicKey.toBase58(), toAddress: btcAddress });
const quote = await fetch(`${BASE}/quote?${q}`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
quote = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, params={
    "from": "SOL", "to": "BTC", "amount": "1", "fromAddress": sol_wallet, "toAddress":
btc_address,
}).json()
```

Response


```
{
  "route": "sol-btc",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "1000000000",
  "toAmount": "113270",
  "toAmountMin": "113270",
  "formatted": {
    "send": "1 SOL",
    "receive": "0.0011327 BTC",
    "pair": "SOL → BTC"
  },
  "expiresAt": 1794535712,
  "expiresAtIso": "2026-11-13T02:08:32.000Z",
  "quoteId": "53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db",
  "order": {
    "route": "sol-btc",
    "orderId": "53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db",
    "fromAddress": "7pvsbHxexZeVgRsUw2kg6tX2939NozKZuvFNR2VkqNrL",
    "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
    "asset": {
      "symbol": "SOL",
      "mint": "11111111111111111111111111111111",
      "decimals": 9
    },
    "amount": "1000000000",
    "quotedSats": "113270",
    "expires": 1794535893,
    "sandbox": "…",
    "quote": {
      "expiresAt": 1794535712,
      "id": "53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db",
      "token": "762249da043c46a1cfb001c6924e034aaccd985debbd861d0bee6ed351618b96",
      "ask":
"from=SOL&to=BTC&amount=1&fromAddress=7pvsbHxexZeVgRsUw2kg6tX2939NozKZuvFNR2VkqNrL&toAddress=bc1qw5
    }
  },
  "issues": {
    "balance": null,
    "allowance": null,
    "orderCap": null
  },
  "steps": [
    {
      "id": "quote",
      "kind": "signature",
```

```
      "description": "Sign this with 7pvsbHxexZeVgRsUw2kg6tX2939NozKZuvFNR2VkqNrL (signMessage
over the UTF-8 text). Sandbox: it authorizes nothing.",
      "signatureKind": "solana-message",
      "message": "Maxy Sandbox order
53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db\nStep: quote\nWallet:
7pvsbHxexZeVgRsUw2kg6tX2939NozKZuvFNR2VkqNrL\nMaxy sandbox. This moves nothing and cannot be
spent.",
      "encoding": "utf8",
      "signer": "7pvsbHxexZeVgRsUw2kg6tX2939NozKZuvFNR2VkqNrL"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and the signature (base64); back comes the deposit
transaction to sign.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    },
    {
      "id": "deposit",
      "kind": "transaction",
      "chain": "solana",
      "description": "Sign the deposit POST /orders returns and POST it to
/orders/{orderId}/steps. Sandbox: it is not sent.",
      "transaction": null,
      "signer": "7pvsbHxexZeVgRsUw2kg6tX2939NozKZuvFNR2VkqNrL"
    }
  ],
  "signing": "…",
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 9d64b57f-0f69-4f96-8123-8756427990d5), with long fields folded.

3. The order. The wallet signs the `quote` step's text (`signMessage` over the UTF-8 message). Send the signature as base64.

curl

```
curl -s -X POST "$BASE/orders" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d "$(jq -c --arg sig "$QUOTE_SIG" '{order: .order, signatures: {quote: $sig}}' quote.json)" >
placed.json
```

TypeScript

```
const message = quote.steps.find((s) => s.id === 'quote').message;
const sig = await wallet.signMessage(new TextEncoder().encode(message));
const placed = await fetch(`${BASE}/orders`, {
  method: 'POST',
  headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' },
  body: JSON.stringify({ order: quote.order, signatures: { quote:
btoa(String.fromCharCode(...sig)) } }),
}).then((r) => r.json());
```

Python

```
placed = requests.post(f"{BASE}/orders", headers={"x-maxy-key": KEY},
                       json={"order": quote["order"], "signatures": {"quote":
quote_signature_b64}}).json()
```

Response


```
{
  "orderId": "53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db",
  "route": "sol-btc",
  "expiresAt": 1794535894,
  "order": "…",
  "status": "/status?
orderId=53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db&route=sol-btc",
  "steps": [
    {
      "id": "deposit",
      "kind": "transaction",
      "chain": "solana",
      "description": "Sign this, then POST it to next. Sandbox: it is not sent.",
      "transaction": "…",
      "encoding": "base64",
      "simulated": true,
      "signer": "7pvsbHxexZeVgRsUw2kg6tX2939NozKZuvFNR2VkqNrL"
    }
  ],
  "next": {
    "method": "POST",
    "path": "/orders/53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db/steps",
    "body": {
      "order": "…",
      "signatures": "…",
      "transaction": "<the signed deposit transaction, base64>"
    }
  },
  "expiresAtIso": "2026-11-13T02:11:34.000Z",
  "refundAt": 1794535894,
  "sandbox": true
}
```


`POST /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` 7bee8414-9b42-48bf-9fb5-7b4c1b8debd5), with long fields folded.

4. The deposit. This step replaces the EVM escrow transaction. The wallet adapter signs `steps[0].transaction`, a base64 Solana transaction, and you post it to `next`. The API sends it.

curl

```
curl -s -X POST "$BASE/orders/$ORDER_ID/steps" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d "$(jq -c --arg tx "$SIGNED_TX" '.next.body + {transaction: $tx}' placed.json)"
```

TypeScript

```
const tx = VersionedTransaction.deserialize(Buffer.from(placed.steps[0].transaction, 'base64'));
const signed = await wallet.signTransaction(tx);
await fetch(`${BASE}${placed.next.path}`, {
  method: 'POST',
  headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' },
  body: JSON.stringify({ ...placed.next.body, transaction:
Buffer.from(signed.serialize()).toString('base64') }),
});
```

Python

```
requests.post(BASE + placed["next"]["path"].removeprefix("/"), headers={"x-maxy-key": KEY},
              json={**placed["next"]["body"], "transaction": signed_tx_b64})
```

Response

```
{
  "orderId": "53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db",
  "route": "sol-btc",
  "submitted": true,
  "signature":
"2wkCi1pKYrWwdYbupVbe7dNZNRYvXefjqot5kMjfyvxZu8wsJ2pQ6KJQynUUpGgVeTz7ZyKnGxrXXvgu5vhF8nvC",
  "txHashes": [
    "2wkCi1pKYrWwdYbupVbe7dNZNRYvXefjqot5kMjfyvxZu8wsJ2pQ6KJQynUUpGgVeTz7ZyKnGxrXXvgu5vhF8nvC"
  ],
  "simulated": true,
  "expiresAt": 1794535896,
  "status": "/status?
orderId=53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db&route=sol-btc",
  "sandbox": true
}
```

`POST /orders/{id}/steps`: a real sandbox answer, 13 November 2026 (`x-request-id` d28192da-1864-4588-90cc-7013f4717174).

5. Watch. As on the EVM buy, with a Solana order id (no `0x`).

Response


```
{
  "orderId": "53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db",
  "route": "sol-btc",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f587656860fc0ac11f0ac7345ca6ce976f5efe7594f1d6ba99e3f",
  "txs": [
    {
      "label": "escrow",
      "hash":
"2wkCi1pKYrWwdYbupVbe7dNZNRYvXefjqot5kMjfyvxZu8wsJ2pQ6KJQynUUpGgVeTz7ZyKnGxrXXvgu5vhF8nvC"
    },
    {
      "label": "btc_payout",
      "hash": "53414e44424f587656860fc0ac11f0ac7345ca6ce976f5efe7594f1d6ba99e3f"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}` : a real sandbox answer, 13 November 2026 (`x-request-id` 6d6e85c0-3e4a-4ed2-8794-75cb914bccb5).

Only a Solana-format wallet can sign these. The SDK takes the provider shape Phantom, Solflare and Backpack inject (signers); there is no published test matrix per wallet.

5.4c States

Phase	Meaning
`OPEN`	The solver's offer is on chain
`PAYING`	The user's escrow is being deposited
`LOCKED`	The SOL is in the program's escrow
`PAID`	The solver paid the BTC; `btcTxId` is set
`SETTLED`	The payment was proven on Solana; the escrow went to the solver

An offer the user never funds expires after 10 minutes. An escrow whose BTC is never proven refunds the user's wallet when it expires.

5.4d Selling BTC for SOL

The other way round, `btc-sol`: the user makes an offer on the settlement program, and Maxy's solver takes it by locking its own SOL there. The user's Bitcoin payment, once proven, releases the SOL to the user. The SDK places it with `swap.createBtcToSolOrder` and settles it with `swap.settleBtcToSol`; the steps are in the swap guide. Over HTTP: BTC to SOL, end to end.

5.4e Differences from EVM buys

- Solana orders are authorized by the wallet's own signatures and go to the solver's Solana settlement service. Over HTTP, `POST /orders` and `POST /orders/{id}/steps` carry them there.
- The SOL is never sent to a Maxy address: it moves only into the settlement program's escrow.

5.4f Errors on this page

Code	Cause	Fix
`quote_expired`	The solver's offer lapsed before the deposit	Quote again
`quote_mismatch`	Selling BTC for SOL: the solver's fresh price is more than 0.1% below the quote	Quote again
`user_rejected`	The wallet declined a signature	Ask again
`signer_unsupported`	The wallet adapter cannot sign what the order needs	Use another wallet (signers)

Next: 5.5 Sell BTC to a chain.

Last verified: 13 November 2026, against the sandbox.

5.5 Sell BTC to a chain

After this page your user can sell BTC for a coin on an EVM chain, with the payout locked for them before they pay.

Before you start: an API key, an EVM wallet that receives the payout, and the 2.1 Quickstart. **Time:** 20 minutes.

5.5a The flow

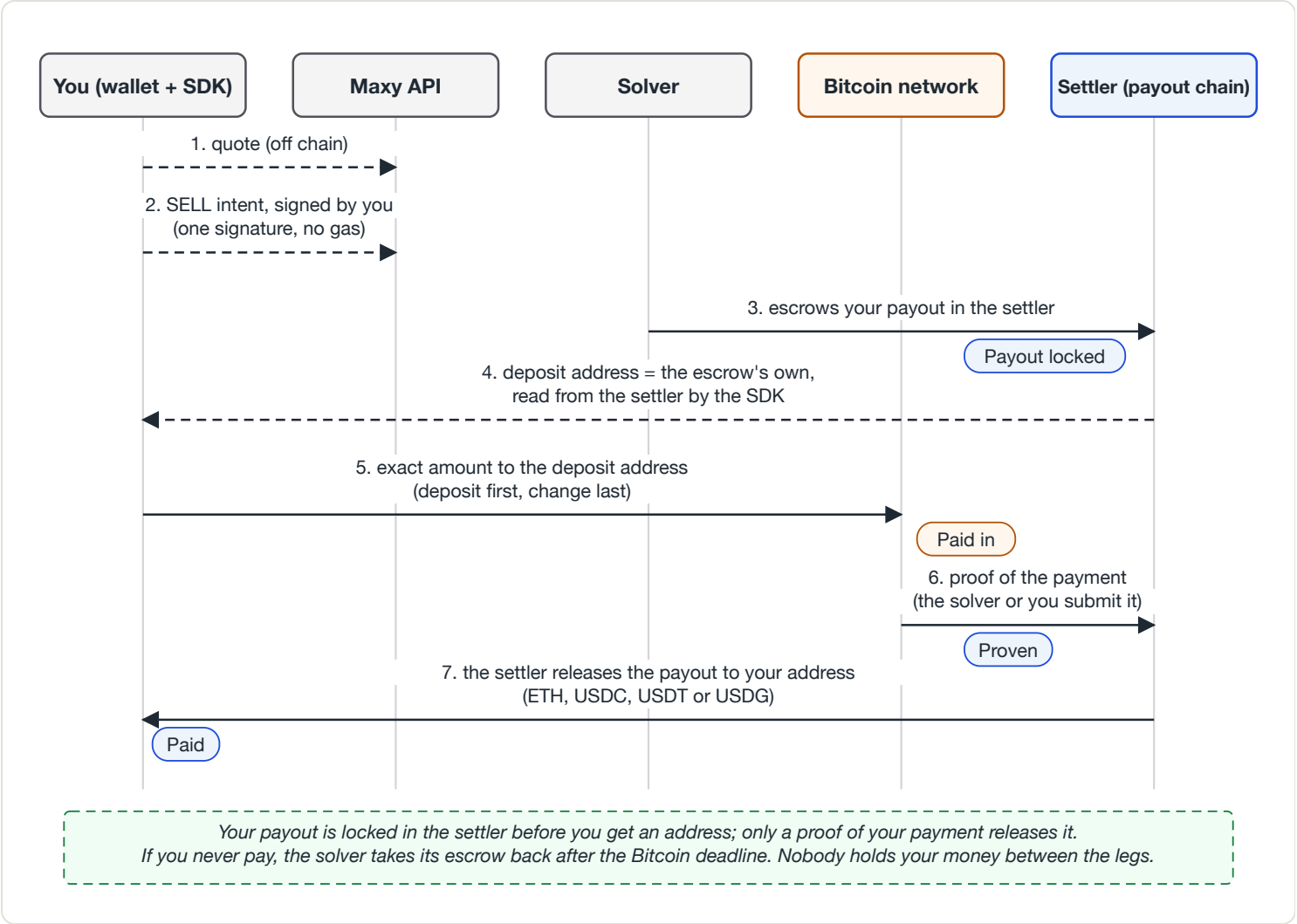


Figure 1. The solver's payout is escrowed in the settler before the user gets an address. The user pays that escrow's own Bitcoin script, and a proof of the payment releases the payout to the user.

1. **Sign in.** Selling acts for a wallet, so it needs a session: `maxy.auth.signIn(evm)` (Sign-In with Ethereum).

2. **Quote.** `swap.quote({ from: BTC, to: USDC on Base, amount: sats })`. The quote is the solver's executable payout.
3. **Sign.** The user's EVM wallet signs one EIP-712 sell intent. No transaction, no gas.
4. **Escrow first.** The solver commits the intent in the settler with a bond: the payout is now locked for the user.
5. **The address.** The SDK reads the commitment back from the contract and returns `order.deposit`: `address`, `sats`, `expiresAt`. The address is decoded from the script stored in the contract, never from server text.
6. **Pay.** The user sends exactly `order.deposit.sats` to `order.deposit.address` in one output, from any wallet, or `swap.payDeposit(order, bitcoinSigner)` pays it.
7. **Release.** After the order's confirmations, a proof of the payment releases the payout to the user. The solver submits it; anyone may, and `swap.claimBtcToEvm` submits it from the user's own wallet.

TypeScript

```
import { createMaxyClient, evmSignerFromWalletClient } from '@maxy-labs/sdk';

let accessToken: string | null = null;
const maxy = createMaxyClient({ apiKey: process.env.MAXY_API_KEY, getAccessToken: () =>
accessToken });
const evm = evmSignerFromWalletClient(walletClient);     // the user's wallet (setup on buy BTC
from an EVM chain)

// Selling acts for a wallet, so it signs the wallet in (Sign-In with Ethereum). A wallet's
first sign-in creates its
// wallet profile and needs a username. The wallet shows "maxybeta.com wants you to sign in"
unless Maxy has
// registered your domain for you (then pass it as `domain`).
const { session } = await maxy.auth.signIn(evm, { username: 'user-1234' });
accessToken = session.accessToken;

const q = await maxy.swap.quote({ from: 'BTC', to: 'USDC on base', amount: 50_000n }); // sats
const order = await maxy.swap.createOrder(q, { evm });    // one signature; the payout goes to
this wallet
console.log(order.deposit);   // { address, sats, expiresAt, verifiedBy: 'settler' }

await maxy.swap.payDeposit(order, bitcoinSigner);         // or show address + sats and let any
wallet pay
const done = await maxy.swap.waitForCompletion(order);    // completed: the settler paid the
user
```

No SDK? 5.6 Sell BTC over HTTP.

5.5b What to show

State	Show
`awaiting_deposit`	"Send exactly 0.0005 BTC (50,000 sats) to bc1q… in one payment, now." with the time left from `order.deposit.expiresAt`, and a QR of `bitcoin:<address>?amount=0.0005`
`in_progress`	"Payment seen. Waiting for 1 confirmation."
`completed`	"Done. 38.01 USDC is in your wallet." with the payout transaction link

5.5c Payment rules

> **Pay once, exactly, on time.** One output of exactly `order.deposit.sats` to `order.deposit.address`: the contract proves only an output of exactly that amount, so paying less or more cannot be paid out. The window is a number of Bitcoin blocks set per order (`order.settler.btcDeadline`; live orders allow 19 blocks, about 3 hours). A fee bump that keeps the exact amount and address is still one payment; never lower or split it.

Anything else, a different amount or a late payment, is on refunds and failures.

`payDeposit` re-reads the escrow first and pays only while it is still open with enough blocks left.

5.5d Limits

- At least about $10 per order; your key's order size cap above.
- Three open sell orders per wallet.
- The payout goes to the signing wallet, or to `destAddress`.

5.5e Errors on this page

Code	Cause	Fix
`auth_required`	No wallet session	`maxy.auth.signIn(evm)` first
`insufficient_liquidity`	"Unfillable right now": the solver cannot pay this size now	A smaller amount, or later

Code	Cause	Fix
`order_cap_exceeded`	Above your key's cap	Order less
`deposit_address_unverified`	The payout is not escrowed for the user in the settler	**Do not pay.** Start again
`quote_expired`	The quote is older than 60 seconds, or the deposit window passed	A new quote; do not pay

Next: 5.6 Sell BTC over HTTP.

Last verified: 5 November 2026, against the SDK source and the API it runs.

5.6 Sell BTC over HTTP

After this page you can sell BTC with plain HTTP calls: no SDK and no sign-in.

Before you start: an API key, an EVM wallet that receives the payout, and 5.5 Sell BTC to a chain.
Time: 15 minutes.

5.6a The same flow over HTTP

Selling over HTTP needs no SDK and no sign-in. The EVM wallet that receives the payout signs one intent, and the deposit address is read from the settler. The samples use `BASE` and `KEY` from the setup. `fromAddress` is any Bitcoin address of the payer, and `toAddress` is the wallet that signs.

1. A price.

curl

```
curl -s "$BASE/price?from=BTC&to=USDC%20on%20base&amount=0.0005"
```

TypeScript

```
const price = await fetch(`${BASE}/price?${new URLSearchParams({ from: 'BTC', to: 'USDC on
base', amount: '0.0005' })}`).then((r) => r.json());
```

Python

```
price = requests.get(f"{BASE}/price", params={"from": "BTC", "to": "USDC on base", "amount":
"0.0005"}).json()
```

Response


```
{
  "route": "btc-evm",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "50000",
  "toAmount": "35372916",
  "toAmountMin": "35372916",
  "formatted": {
    "send": "0.0005 BTC",
    "receive": "35.37 USDC on Base",
    "pair": "BTC → USDC on Base"
  },
  "expiresAt": 1794535534,
  "expiresAtIso": "2026-11-13T02:05:34.000Z",
  "sandbox": true
}
```


`GET /price`: a real sandbox answer, 13 November 2026 (`x-request-id` 631d8e8f-ffcd-47de-a50e-3fd670c6d19a), with long fields folded.

2. A quote.

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" -d from=BTC --data-urlencode "to=USDC on
base" -d amount=0.0005 \
  -d fromAddress="$MY_BTC_ADDRESS" -d toAddress="$WALLET" > quote.json
```

TypeScript

```
const q = new URLSearchParams({ from: 'BTC', to: 'USDC on base', amount: '0.0005', fromAddress:
myBtcAddress, toAddress: account.address });
const quote = await fetch(`${BASE}/quote?${q}`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
quote = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, params={
    "from": "BTC", "to": "USDC on base", "amount": "0.0005", "fromAddress": my_btc_address,
"toAddress": account.address,
}).json()
```

Response


```
{
  "route": "btc-evm",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "50000",
  "toAmount": "35320482",
  "toAmountMin": "35320482",
  "formatted": {
    "send": "0.0005 BTC",
    "receive": "35.32 USDC on Base",
    "pair": "BTC → USDC on Base"
  },
  "expiresAt": 1794535681,
  "expiresAtIso": "2026-11-13T02:08:01.000Z",
  "quoteId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "order": {
    "route": "btc-evm",
    "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
    "chainId": 8453,
    "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
    "satAmount": 50000,
    "maker": "0x11B172F50AaD19f0372c3de1d2850cA1Cd180064",
    "intent": "…",
    "scriptPubKey": "0x",
    "receiveIndex": null,
    "payerBtcAddress": "bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq",
    "quotedRaw": "35320482",
    "settlementAmount": "35320482",
    "lockBy": 1794536221,
    "status": "/status?
orderId=0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926&chainId=8453&route=btc-
evm",
    "sandbox": "…",
    "quote": {
      "expiresAt": 1794535681,
      "id": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
      "token": "4718ecf9ac1b6808582ffeef515c6ee59c761d0afce3e0a3e4d8a6096565b5fe",
      "ask":
"from=BTC&to=USDC+on+base&amount=0.0005&fromAddress=bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq&toAd
    }
  },
  "issues": {
    "balance": null,
    "allowance": null,
    "orderCap": null
  },
```

```
  "steps": [
    {
      "id": "intent",
      "kind": "signature",
      "description": "Sign the sell order (EIP-712) with
0x11B172F50AaD19f0372c3de1d2850cA1Cd180064. No gas. Sandbox: this signature authorizes
nothing.",
      "signatureKind": "eip712",
      "typedData": {
        "domain": {
          "name": "Maxy Sandbox",
          "version": "1",
          "chainId": 8453,
          "verifyingContract": "0x0000000000000000000000000000000000000000"
        },
        "types": "…",
        "primaryType": "MaxySandboxOrder",
        "message": {
          "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
          "step": "intent",
          "notice": "Maxy sandbox. This moves nothing and cannot be spent."
        }
      },
      "hash": "0x8ff83c730344e3da0242361abf5a2007f3b653649a7f18f3aa3ce1355bcc633d"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signature. In the sandbox nothing is locked.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    },
    {
      "id": "deposit",
      "kind": "deposit",
      "description": "After POST /orders: GET /orders/{orderId}/deposit until it answers the
address. Sandbox: pay nothing; the order walks by itself.",
      "method": "GET",
      "path":
"/orders/0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926/deposit?
chainId=8453&maker=0x11B172F50AaD19f0372c3de1d2850cA1Cd180064&token=0x833589fCD6eDb6E08f4c7C32D4f71
    }
  ],
  "signing": "…",
  "sandbox": true
}
```

`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 7afe1501-75a8-46de-8761-3f0b30f7091f), with long fields folded.

3. Sign and order. Sign the `intent` step as on the EVM buy, then post it. The solver locks the payout in the settler before you get an address.

curl

```
curl -s -X POST "$BASE/orders" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d "$(jq -c --arg sig "$INTENT_SIG" '{order: .order, signatures: {intent: $sig}}' quote.json)"
> placed.json
```

TypeScript

```
const placed = await fetch(`${BASE}/orders`, {
  method: 'POST',
  headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' },
  body: JSON.stringify({ order: quote.order, signatures }),
}).then((r) => r.json());
```

Python

```
placed = requests.post(f"{BASE}/orders", headers={"x-maxy-key": KEY},
                       json={"order": quote["order"], "signatures": signatures}).json()
```

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "chainId": 8453,
  "escrow": "locked",
  "commitTx": "0x53414e44424f584b58e247ef5b78f06ed6ee95982a04989100da7f0477ce5037",
  "lockBy": 1794536221,
  "status": "/status?
orderId=0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926&chainId=8453&route=btc-
evm",
  "next": {
    "method": "GET",
    "path": "/orders/0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926/deposit?
chainId=8453&maker=0x11B172F50AaD19f0372c3de1d2850cA1Cd180064&token=0x833589fCD6eDb6E08f4c7C32D4f71
    "description": "Poll until it answers ready: true. Sandbox: pay nothing; the order walks by
itself."
  },
  "sandbox": true
}
```

`POST /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` 28987b1d-614c-4fbc-ade4-1663a8bcaa96).

4. The payment instructions. Poll `next.path` until `ready` is `true`. `bitcoinUri` is the BIP-21 URI for the QR code.

curl

```
curl -s "$BASE$(jq -r '.next.path | ltrimstr("/")' placed.json)"
```

TypeScript

```
const deposit = await fetch(BASE + placed.next.path.replace('/', '')).then((r) => r.json());
```

Python

```
deposit = requests.get(BASE + placed["next"]["path"].removeprefix("/")).json()
```

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "chainId": 8453,
  "settler": "0x0000000000000000000000000000000000000000",
  "state": "NONE",
  "ready": false,
  "reason": "The solver is locking the payout in the settler. Do not send BTC. Ask again in a
few seconds.",
  "retryAfterSeconds": 3,
  "lockBy": 1794536221,
  "sandbox": true
}
```


`GET /orders/{id}/deposit`: a real sandbox answer, 13 November 2026 (`x-request-id` 736778d4-155b-49cf-97be-df75d15b0af7).

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "chainId": 8453,
  "settler": "0x0000000000000000000000000000000000000000",
  "state": "COMMITTED",
  "ready": true,
  "address": "tb1qwrdwhtmx7snnesh4k9vt82ggn9xmyg8zwetx06",
  "sats": 50000,
  "amountBtc": "0.0005",
  "bitcoinUri": "bitcoin:tb1qwrdwhtmx7snnesh4k9vt82ggn9xmyg8zwetx06?amount=0.0005",
  "scriptPubKey": "0x001470daebaf66f4273cc2f5b158b3a908994db220e2",
  "payout": {
    "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
    "amount": "35320482",
    "to": "0x11B172F50AaD19f0372c3de1d2850cA1Cd180064"
  },
  "expiresAt": 1794539231,
  "btcDeadline": 0,
  "blocksLeft": 6,
  "verifierHeight": 0,
  "requiredConfirmations": 1,
  "intentHash": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "checked": "order",
  "simulated": true,
  "instructions": "Sandbox: do not pay. This is a testnet address, and the order moves on by
itself.",
  "verify": "…",
  "claim": "/orders/0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926/claim?
chainId=8453",
  "status": "/status?
orderId=0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926&chainId=8453&route=btc-
evm",
  "sandbox": true
}
```


`GET /orders/{id}/deposit`: a real sandbox answer, 13 November 2026 (`x-request-id` 58d489ce-b64a-4f0c-9220-6e350066ada1), with long fields folded. Sandbox: the address is a `tb1q`, a mainnet wallet refuses it, and the settler is the zero address. Pay nothing.

5. The states. `awaiting_deposit`, then `in_progress` once a payment is seen, then `completed`:

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "state": "awaiting_deposit",
  "rawStatus": "SANDBOX_AWAITING_DEPOSIT",
  "isComplete": false,
  "btcTxId": null,
  "txs": [],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "refundAt": null,
  "refundAtIso": null,
  "refundAfterBlock": null,
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 96078ee5-5b05-4914-a0af-f1ed673794b2).

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f58f9ac0e49ea608e743552bca7a46625f415623b2b7c2144d436",
  "txs": [
    {
      "label": "btc_payment",
      "hash": "53414e44424f58f9ac0e49ea608e743552bca7a46625f415623b2b7c2144d436"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` b7ca0acb-6ef2-4f2e-adbd-b0549339c87c).

6. Claim, if the solver is slow. `GET /orders/{id}/claim` answers the proof as a transaction any wallet may send. The payout still goes only to the user.

curl

```
curl -s "$BASE/orders/$ORDER_ID/claim?chainId=8453"
```

TypeScript

```
const claim = await fetch(`${BASE}/orders/${orderId}/claim?chainId=8453`).then((r) => r.json());
```

Python

```
claim = requests.get(f"{BASE}/orders/{order_id}/claim", params={"chainId": 8453}).json()
```

Response

```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "chainId": 8453,
  "claimable": false,
  "reason": "No payment seen yet. Pay first, or wait for the payment to confirm.",
  "transaction": null,
  "state": "COMMITTED",
  "retryable": true,
  "sandbox": true
}
```

`GET /orders/{id}/claim`: a real sandbox answer, 13 November 2026 (`x-request-id` 25c117d3-62be-4a43-8762-f865f7c8ce1f).

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "chainId": 8453,
  "claimable": true,
  "function": "fillIntentWithFeeToken",
  "explanation": "Proves the BTC payment to the settler, which releases the payout to the maker.
Sandbox: nothing to send; the order completes by itself.",
  "btcTxid": "53414e44424f58f9ac0e49ea608e743552bca7a46625f415623b2b7c2144d436",
  "btcBlockHeight": 0,
  "payout": {
    "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
    "amount": "35320482",
    "to": "0x11B172F50AaD19f0372c3de1d2850cA1Cd180064"
  },
  "transaction": {
    "chainId": 8453,
    "from": "0x11B172F50AaD19f0372c3de1d2850cA1Cd180064",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f58282aa26e0d84fb2cf2505b5e6edeafa1f43680b552e1e2ed45",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": {
    "ok": true
  },
  "sandbox": true
}
```


`GET /orders/{id}/claim`: a real sandbox answer, 13 November 2026 (`x-request-id` a7477538-3026-40e7-a475-3f633477e530).

5.6b Errors on this page

Code	HTTP	Cause	Fix
`order_cap_exceeded`	422	Above your key's cap	Order less

Code	HTTP	Cause	Fix
`insufficient_liquidity`	422	The solver cannot pay this size now	A smaller amount, or later
`quote_expired`	409	The quote is older than 60 seconds, or the deposit window passed	A new quote; do not pay

Next: 5.7 Track orders and receive webhooks.

Last verified: 13 November 2026, against the sandbox.

5.7 Track orders and receive webhooks

After this page your system hears of every order change within seconds, verifies each delivery, and finds any order by hash.

Before you start: an API key and an HTTPS endpoint you control. **Time:** 20 minutes.

5.7a Polling or webhooks

	Webhooks	**Polling `GET /status`**
Use for	Your ledger, notifications, support tools	A screen the user is watching right now; a one-off lookup
Latency	Changes are picked up within about 15 seconds	As often as you poll (every 2 seconds is fine while a user watches)
Cost	One delivery per change	One call per poll, against your rate limit

Use both: webhooks for the record, polling while the user watches.

5.7b Register an endpoint

The answer carries the endpoint's signing secret, shown this once: store it in your secret store.

The samples use `BASE` and `KEY` from the setup.

curl

```
curl -s -X POST "$BASE/webhooks" \
  -H "x-maxy-key: $MAXY_API_KEY" -H "content-type: application/json" \
  -d '{ "url": "https://your.app/maxy/webhooks" }'
```

TypeScript

```
const res = await fetch(`${BASE}/webhooks`, {
  method: 'POST',
  headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' },
  body: JSON.stringify({ url: 'https://your.app/maxy/webhooks' }),
});
const { id, secret } = await res.json(); // store secret: it is shown once
```

Python

```
r = requests.post(f"{BASE}/webhooks", json={"url": "https://your.app/maxy/webhooks"}, headers=
{"x-maxy-key": KEY})
hook = r.json()  # hook["secret"] is shown once: store it
```

Response

```
{
  "id": "0d625e15-f920-4f77-a044-f4a9ddd040c6",
  "url": "https://example.com/maxy/webhooks",
  "events": ["order.updated"],
  "secret": "whsec_… (shown once: store it)",
  "createdAt": "2026-11-02T15:59:02.376347+00:00"
}
```

`POST /webhooks` (HTTP 201): a real answer from the live API, 2 November 2026 (`x-request-id` a607855e-bffe-4fc8-a9e9-8e8b6d2a5703); the secret is masked here.

Orders placed through the HTTP API, and orders the SDK places with your key, are followed automatically. Any other order is followed after `POST /orders/watch` with its `orderId`.

5.7c What arrives

A `POST` with a JSON body, sent when an order's `state` or its `btcTxId` changes. This is the body of the published test vector:

Response


```
{
  "id": "evt_92c8c63c194b601ce42da1a3",
  "type": "order.updated",
  "createdAt": "2026-11-02T18:29:57.000Z",
  "data": {
    "orderId": "0xed3985cf8b6dcc9a30448c9faaefbaf3a2f2ef9c06f50ce1f46dd955212ce2ea",
    "route": "evm-btc-gasless",
    "chainId": 8453,
    "state": "completed",
    "previousState": "in_progress",
    "rawStatus": "SETTLED",
    "isComplete": true,
    "btcTxId": "6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea3f",
    "txs": [
      {
        "label": "escrow",
        "hash": "0xf103819940fef19e88f9c0eafb24a52bad8de86c277e18e8e2e60a8594c7f8d1"
      },
      {
        "label": "btc_payout",
        "hash": "6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea3f"
      },
      {
        "label": "settlement",
        "hash": "0xef66f624a34cff33a5d5660e1666e15669bc02dfc8b0a27fa827e8b268e702f2"
      }
    ]
  }
}
```


Event	When it fires
`order.updated`	An order's `state` or `btcTxId` changed. `data` is the order's status, as `GET /status` returns it, plus `previousState`
`webhook.test`	You called `POST /webhooks/{id}/test`. `data` is `{ "webhookId": "…", "message": "A test event from Maxy. Verify its signature with your endpoint secret." }`

Every polled `state` has this one webhook equivalent. The full schemas are on webhook events.

5.7d Verify the signature

Every delivery carries three headers:

Header	Value
`Maxy-Signature`	`t=<unix seconds>,v1=<hex>`: `v1` is HMAC-SHA256 of `<t>.<raw body>`, keyed with the endpoint secret
`Maxy-Event-Id`	The event `id`, the same as in the body
`Maxy-Event-Type`	`order.updated` or `webhook.test`

Verify the raw body before you parse it, compare in constant time, and reject a `t` more than 5 minutes from now.

TypeScript

```
import express from 'express';
import { verifyWebhook, type MaxyWebhookEvent } from '@maxy-labs/sdk';

const app = express();
app.post('/maxy/webhooks', express.raw({ type: 'application/json' }), async (req, res) => {
  let event: MaxyWebhookEvent;
  try {
    event = await verifyWebhook(req.body.toString(), req.header('maxy-signature'),
process.env.MAXY_WEBHOOK_SECRET!);
  } catch {
    return res.sendStatus(400); // a bad or stale signature: not from Maxy, or a replay
  }
  // Your own store: deliveries repeat, so dedupe on the event id.
  if (await alreadyProcessed(event.id)) return res.sendStatus(200);
  if (event.type === 'order.updated') await recordOrderState(event.data.orderId,
event.data.state, event.data.btcTxId);
  res.sendStatus(200);
});
```

Python

```
import hmac, hashlib, time

def verify(raw_body: bytes, header: str, secret: str, tolerance=300, now=None) -> bool:
    parts = dict(p.strip().split("=", 1) for p in header.split(","))
    t, sig = int(parts["t"]), parts["v1"]
    expected = hmac.new(secret.encode(), f"{t}.".encode() + raw_body,
hashlib.sha256).hexdigest()
    now = time.time() if now is None else now
    return hmac.compare_digest(expected, sig) and abs(now - t) <= tolerance
```

Test vector. A delivery signed with the SDK's own signer and a published test secret. Check your verifier at the vector's own time (`now=` its `t`; `nowSec` with `verifyWebhook`): it must accept the

vector, refuse it with one byte of the body changed, and refuse it at `t + 301`. The kit's `verify-webhook.mjs` and `verify_webhook.py` check all three:

Response


```
{
  "secret": "whsec_maxy_test_vector_not_a_real_secret_000000",
  "timestamp": 1793644197,
  "body": "
{\"id\":\"evt_92c8c63c194b601ce42da1a3\",\"type\":\"order.updated\",\"createdAt\":\"2026-11-
02T18:29:57.000Z\",\"data\":
{\"orderId\":\"0xed3985cf8b6dcc9a30448c9faaefbaf3a2f2ef9c06f50ce1f46dd955212ce2ea\",\"route\":\"evm
btc-
gasless\",\"chainId\":8453,\"state\":\"completed\",\"previousState\":\"in_progress\",\"rawStatus\":
[{\"label\":\"escrow\",\"hash\":\"0xf103819940fef19e88f9c0eafb24a52bad8de86c277e18e8e2e60a8594c7f8d
{\"label\":\"btc_payout\",\"hash\":\"6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea
{\"label\":\"settlement\",\"hash\":\"0xef66f624a34cff33a5d5660e1666e15669bc02dfc8b0a27fa827e8b268e7
  "headers": {
    "content-type": "application/json",
    "user-agent": "Maxy-Webhooks/1",
    "maxy-event-id": "evt_92c8c63c194b601ce42da1a3",
    "maxy-event-type": "order.updated",
    "maxy-signature":
"t=1793644197,v1=3dcfb13254225870c0d80924d3a267d926286c09d26fb054358bb793361aaf68"
  }
}
```

5.7e Delivery

- Answer with any `2xx` within 10 seconds.
- Anything else is retried 8 times over about 23 hours (30 s, 2 min, 10 min, 30 min, 1 h, 3 h, 6 h, 12 h).
- Delivery is at least once: dedupe on the event `id`.
- Up to 8 events go out at once, so two events for one order can arrive out of order. Order them by `createdAt`, or ignore an event whose `previousState` is not the state you hold.
- An order is followed until it reaches a final state, or for 3 days. A refund sent after 3 days sends no webhook; `GET /status` always has the current state.
- `POST /webhooks/{id}/events/{eventId}/replay` sends a past event again, now, with a new signature and the same event id. There is no dashboard.
- `GET /webhooks/{id}/events` shows the last 25 `order.updated` events for an endpoint, with attempts and your endpoint's last answer (test events are not listed). Events are kept for 30 days. After an outage on your side, read each open order's current state from `GET /status`: it is always the truth.

Method and path	What it does
`POST /webhooks`	Register an endpoint (`{ url }`, HTTPS on a public host; up to 5 per key)
`GET /webhooks`	Your endpoints (never the secrets)
`DELETE /webhooks/{id}`	Stop sending to an endpoint
`POST /webhooks/{id}/test`	Send a signed `webhook.test` event now; answers with your endpoint's HTTP status
`GET /webhooks/{id}/events`	The last 25 events for an endpoint. `?signed=1` adds each one's raw body and a signature made now
`POST /webhooks/{id}/events/{eventId}/replay`	Send a past event again, with the same event id
`POST /orders/watch`	Follow any order: `{ orderId, chainId?, route?, toChainId? }`

5.7f Local development with no tunnel

`maxy webhooks listen --forward-to http://localhost:3000/maxy/webhooks` gives your laptop real signed events with nothing exposed. It prints the secret to verify with, and removes its endpoint when you stop it.

5.7g Errors on this page

Code	HTTP	Cause	Fix
`invalid_input`	400	The webhook URL is not HTTPS on a public host; a `txHash` that is not 32 bytes	Fix the input
`not_found`	404	No order has that transaction, or no webhook has that id for your key	Check the hash; retry a new one after a few seconds
`conflict`	409	A sixth endpoint: a key has at most 5	Delete one first (`DELETE /webhooks/{id}`)
`auth_required`	401	Webhook routes need your key	Send `x-maxy-key`

Next: 5.8 Streams and lookup by hash.

Last verified: 5 November 2026, against the sandbox.

5.8 Streams and lookup by hash

After this page you can follow an order as a stream, and find any order from a transaction hash.

Before you start: an order id or a transaction hash. **Time:** 3 minutes.

5.8a A stream instead of polling

`GET /orders/{id}/events/stream` sends one `order.updated` per change, until the order ends. `GET /prices/stream` answers `GET /price` every five seconds or more (`intervalSeconds`). A connection counts as one request and lasts at most two minutes; its last event says to reconnect. `orders.watch` in the SDK does this and falls back to polling.

curl

```
curl -sN -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/events/stream?chainId=8453"
```

TypeScript

```
for await (const state of maxy.orders.watch(orderId, { chainId: 8453 })) render(state);
const stop = maxy.prices.stream({ from: 'ETH on base', to: 'BTC', amount: '0.01' }, (price) =>
show(price.formatted));
```

Python

```
for state in maxy.orders.watch(order_id, chain_id=8453):  # Python polls: one second, backing
off to five
    render(state)
```

A stream's first event, as it arrives:

```
retry: 5000

event: price
data: {"route":"evm-btc-escrow","liquidityAvailable":true,"from":
{"chain":8453,"chainName":"Base","token":"0x0000000000000000000000000000000000000000","symbol":"ETH
{"chain":"bitcoin","chainName":"Bitcoin","token":"BTC","symbol":"BTC","decimals":8,"caip2":"bip122:
{"send":"0.004 ETH on Base","receive":"0.00010868 BTC","pair":"ETH on Base →
BTC"},"expiresAt":1794536203,"expiresAtIso":"2026-11-13T02:16:43.000Z","limits":
{"minSats":546},"sandbox":true}
```

`GET /prices/stream`: a real sandbox answer, 13 November 2026 (`x-request-id` 744113cb-861c-477c-940d-adb3be1a2cd2).

5.8b In a browser: a stream ticket

`EventSource` cannot send a header, and every route takes a key. So a page asks a ticket with its key first (a publishable key works, from its own origins), then opens the stream with `?ticket=` and no key. `POST /orders/{id}/stream-ticket` answers `{ ticket, expiresAt, stream }`: `stream` is the URL to give `EventSource`. The ticket opens that one order's stream for 10 minutes; `POST /prices/stream-ticket?from=…&to=…` does the same for one pair's prices, at any amount. A ticket carries the key's id, never the key; another order, another pair, a changed or an expired ticket answers `401 invalid_ticket`, and a secret key put in a URL answers `400`. `maxy.orders.watch` in a browser does all of this for you; on a server it keeps the header.

TypeScript

```
const { stream } = await fetch(`${BASE}/orders/${orderId}/stream-ticket?chainId=8453`, { method:
'POST', headers: { 'x-maxy-key': PUBLISHABLE_KEY } }).then((r) => r.json());
const events = new EventSource(stream);
events.addEventListener('order.updated', (e) => render(JSON.parse(e.data)));
events.addEventListener('end', () => events.close());
```

`GET /orders/{id}/events` is the same history after the fact: each state with its time, recorded from when Maxy first saw the order.

Response


```
{
  "orderId": "0x53414e44424f5823936eadc65c68429f1fc465d671b4382365c5ff83ffbf13ae",
  "events": [
    {
      "state": "pending",
      "previousState": null,
      "rawStatus": "SANDBOX_PENDING",
      "btcTxId": null,
      "txs": [],
      "at": "2026-11-13T02:09:03.000Z"
    },
    {
      "state": "in_progress",
      "previousState": "pending",
      "rawStatus": "SANDBOX_FILLING",
      "btcTxId": null,
      "txs": [],
      "at": "2026-11-13T02:08:54.000Z"
    },
    {
      "state": "completed",
      "previousState": "in_progress",
      "rawStatus": "SANDBOX_COMPLETED",
      "btcTxId": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7",
      "txs": [
        {
          "label": "btc_payout",
          "hash": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7"
        }
      ],
      "at": "2026-11-13T02:09:06.000Z"
    }
  ],
  "nextCursor": null,
  "sandbox": true
}
```


`GET /orders/{id}/events`: a real sandbox answer, 13 November 2026 (`x-request-id` 38d7829b-fba3-4d13-9e17-26e684d7594d).

5.8c Status by transaction hash

`GET /status?txHash=` finds an order from its escrow transaction or its Bitcoin payment, with your key.

curl

```
curl -s "$BASE/status?txHash=0xf103819940fef19e88f9c0eafb24a52bad8de86c277e18e8e2e60a8594c7f8d1"
# any transaction of the order
```

Response


```
{
  "orderId": "0xed3985cf8b6dcc9a30448c9faaefbaf3a2f2ef9c06f50ce1f46dd955212ce2ea",
  "route": "evm-btc-gasless",
  "state": "completed",
  "rawStatus": "SETTLED",
  "isComplete": true,
  "btcTxId": "6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea3f",
  "txs": [
    {
      "label": "escrow",
      "hash": "0xf103819940fef19e88f9c0eafb24a52bad8de86c277e18e8e2e60a8594c7f8d1"
    },
    {
      "label": "btc_payout",
      "hash": "6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea3f"
    },
    {
      "label": "settlement",
      "hash": "0xef66f624a34cff33a5d5660e1666e15669bc02dfc8b0a27fa827e8b268e702f2"
    }
  ]
}
```


`GET /status?txHash=` with the order's escrow transaction: a real answer from the live API, 2 November 2026 (`x-request-id` 97296e43-aefe-40b2-ab86-3ed695dc1b7b).

5.8d Errors on this page

Code	HTTP	Cause	Fix
`invalid_input`	400	A `txHash` that is not 32 bytes	Fix the hash
`not_found`	404	No order has that transaction: "No Maxy order has that transaction. It may be too new to be recorded; try again in a few seconds."	Check the hash; retry a new one

Next: 5.9 Reconcile and export orders.

Last verified: 13 November 2026, against the sandbox.

5.9 Reconcile and export orders

After this page your finance team can export every order placed with your key and match it to your ledger and the chain.

Before you start: an API key, and orders placed with it. **Time:** 10 minutes.

5.9a The order list

`GET /orders` lists the orders placed with your key, newest first. `from` and `to` take a date (`2026-11-01`) or an ISO 8601 time; `format=csv` returns a spreadsheet. The samples use `BASE` and `KEY` from the setup.

- A page is up to 100 orders (`limit`), with `nextCursor`: pass it as `cursor`, and stop when it is `null`. A CSV export takes `limit` up to 1,000.
- Narrow it with `state`, `wallet`, `externalId` or `externalCustomerId`. Your own ids go on `POST /orders`: up to 128 characters, no spaces. Maxy stores them as given and reads them for nothing else.
- An order stays listed for 30 days after it reaches a final state, so export each month within 30 days.
- Maxy follows an order for 3 days; after that, `state` is the last one recorded and `GET /status` has the current one.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" \
  "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/orders?from=2026-11-01&to=2026-12-01"
```

TypeScript

```
const res = await fetch(`${BASE}/orders?from=2026-11-01&to=2026-12-01`, { headers: { 'x-maxy-key': KEY } });
const { orders } = await res.json();
```

Python

```
r = requests.get(f"{BASE}/orders", params={"from": "2026-11-01", "to": "2026-12-01"}, headers={"x-maxy-key": KEY})
orders = r.json()["orders"]
```

Response


```
{
  "orders": [
    {
      "orderId": "0xf3e3ce420a1585a49cd7d180b967daef6313a47324bf654efdf62c6107accfa8",
      "route": "evm-btc-escrow",
      "chainId": 8453,
      "state": "pending",
      "btcTxId": null,
      "createdAt": "2026-11-02T16:03:02.929002+00:00",
      "doneAt": null
    },
    {
      "orderId": "0x134efd8156d572c0c10754376c8c8fd57178cc34e4d9118651fd824fe530f616",
      "route": "evm-btc-escrow",
      "chainId": 8453,
      "state": "pending",
      "btcTxId": null,
      "createdAt": "2026-11-02T15:58:50.384008+00:00",
      "doneAt": null
    }
  ]
}
```


`GET /orders`: a real answer from the live API, 2 November 2026 (`x-request-id` 92a644e6-9f44-4d27-b2c1-0e443200933f); the first two rows.


```
orderId,route,chainId,state,btcTxId,createdAt,doneAt
0xf3e3ce420a1585a49cd7d180b967daef6313a47324bf654efdf62c6107accfa8,evm-btc-
escrow,8453,pending,,2026-11-02T16:03:02.929002+00:00,
0x134efd8156d572c0c10754376c8c8fd57178cc34e4d9118651fd824fe530f616,evm-btc-
escrow,8453,pending,,2026-11-02T15:58:50.384008+00:00,
0xdf276672bf464934d9ff3f41d3db81fc9f22f027039495956b8565d46e3e0fd3,evm-btc-
escrow,8453,pending,,2026-11-02T15:58:47.429749+00:00,
```


The same list with `format=csv`.

Column	Meaning
`orderId`	The order id: the intent hash for EVM to BTC buys, the same on chain
`route`	`evm-btc-escrow`, `evm-btc-gasless`, … (order lifecycle)
`chainId`	The EVM chain the order was paid on
`state`	The last state Maxy recorded for it (`pending` until Maxy first reads a new order, within a minute)
`btcTxId`	The Bitcoin transaction that paid the user, once sent

Column	Meaning
`createdAt`, `doneAt`	When the order was placed and when it reached a final state (UTC, ISO 8601)

Orders placed through the HTTP API are listed automatically. Orders placed with the SDK are listed once the SDK files them under your key, after `createOrder`. Add an order placed some other way with `POST /orders/watch` and its `orderId`.

5.9b Your own ids and pages

Send your invoice and customer ids with the order, and read the order back by them:

curl

```
curl -s -X POST "$BASE/orders" -H "x-maxy-key: $MAXY_API_KEY" -H "content-type:
application/json" \
  -d "{\"order\": $(jq .order quote.json), \"signatures\": $SIGNATURES, \"externalId\": \"inv-
1042\", \"externalCustomerId\": \"cust-77\"}"
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders?externalId=inv-1042"
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders?limit=100&cursor=$NEXT"
```

TypeScript

```
const order = await maxy.orders.create({ quote, signature, externalId: 'inv-1042',
externalCustomerId: 'cust-77' });
const { data } = await maxy.orders.list({ externalId: 'inv-1042' });
for await (const row of maxy.orders.iterate({ from: '2026-11-01', to: '2026-12-01' })) await
ledger.add(row); // every page
```

Python

```
order = maxy.orders.create(quote["order"], signature=signature, external_id="inv-1042",
external_customer_id="cust-77")
rows = maxy.orders.list(external_id="inv-1042")["orders"]
for row in maxy.orders.iterate(from_="2026-11-01", to="2026-12-01"):
    ledger.add(row)
```

Response


```
{
  "orders": [
    {
      "orderId": "0x53414e44424f5823936eadc65c68429f1fc465d671b4382365c5ff83ffbf13ae",
      "route": "evm-btc-escrow",
      "chainId": 8453,
      "state": "completed",
      "btcTxId": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7",
      "createdAt": "2026-11-13T02:09:03.596028+00:00",
      "doneAt": "2026-11-13T02:09:14.000Z",
      "externalId": "inv-1042",
      "externalCustomerId": "cust-77",
      "wallet": "0xf46351945338d460e6161062Bf833C2D49f4c913"
    },
    {
      "orderId": "0x53414e44424f5832530d812efbf92d239a8118a8b5cbe4a453284774c71436fc",
      "route": "evm-btc-escrow",
      "chainId": 8453,
      "state": "completed",
      "btcTxId": "53414e44424f58040de5a0e657d1a03fb18529a28755837eea1ace7710dcb939",
      "createdAt": "2026-11-13T01:59:09.679363+00:00",
      "doneAt": "2026-11-13T02:00:04.107+00:00",
      "externalId": "inv-1042",
      "externalCustomerId": "cust-77",
      "wallet": "0x010C4e829DDA51b3d25BF986E1Dc7bAA9C048096"
    }
  ],
  "nextCursor": null,
  "sandbox": true
}
```


`GET /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` 97d8fe7b-1886-4e35-aaad-a1362a72c950).

5.9c Month-end, step by step

1. Export the month: `GET /orders?from=2026-11-01&to=2026-12-01&format=csv`.
2. Join it to your own orders on `orderId` (store it when the order is placed; webhooks carry it too).
3. For every row in `completed`: `btcTxId` is the payment to your user. Its amount is on chain (`https://mempool.space/tx/<btcTxId>`) and matches the `toAmount` you showed.
4. For every row in `refunded`, `refundable` or `failed`: no BTC was paid; the user's coins went back to them, or can.

5. Rows in your ledger missing from the export: look each one up with `GET /status?txHash=` (the escrow transaction hash). An order the API has no record of was not placed with your key.

A worked month-end with three sandbox orders, one completed, one refunded and one expired:

`orderId`	`route`	`state`	`btcTxId`	What your ledger does
`0x53414e44424f589a…`	`evm-btc-escrow`	`completed`	`53414e44424f581801…`	Match `btcTxId` to the BTC you showed; book it as filled
`0x53414e44424f58ec…`	`evm-btc-escrow`	`refunded`		No BTC was paid; the coins went back. Book nothing, keep the refund hash
`0x53414e44424f58c5…`	`evm-btc-gasless`	`expired`		Nothing was sent. Close your order

Three real sandbox orders, 13 November 2026, as `GET /orders?format=csv` lists them (`x-request-id` 277ace30-21de-4e33-bbd1-4f1b3950d810).

A month's statement is `GET /statements?month=2026-11&format=json|csv|pdf`: orders with their totals, as the export reads them. The export is the record. The amounts of each order are fixed by the intent the user signed, and both legs are public on chain: the escrow on the paying chain, the payment on Bitcoin.

5.9d Errors on this page

Code	HTTP	Cause	Fix
`auth_required`	401	The order list needs your key	Send `x-maxy-key`
`invalid_input`	400	`from` or `to` is not a date or ISO 8601 time	`2026-11-01`, or `2026-11-01T00:00:00Z`

Next: 5.10 Refunds and failures.

Last verified: 13 November 2026, against the sandbox.

5.10 Refunds and failures

After this page your support team can say where a failed order's money is, who acts next, when, and what to tell the user.

Before you start: 3.1 Order lifecycle. **Time:** 10 minutes.

5.10a The rule

Buying BTC, the user's coins are in their own wallet or in the settler on their own chain. The settler pays the solver only against a proof that the BTC reached the user's address. Every other exit sends them back to the user, and no Maxy server can change that.

5.10b Every failure, in one table

What happened	`state`	Where the money is	Who acts	When	Show the user
The user rejected a signature	(no order)	the user's wallet	nobody		Nothing; back to the confirm screen
Gasless: the solver never opened the escrow	`pending`, then `refundable`	the user's wallet: the Permit2 signature lapses at the intent's deadline, 2 hours after signing (`planRefund` answers `not_refundable`: there is no escrow)	nobody		"Nothing was taken from your wallet."
Escrow open, no solver committed	`pending`, then `refundable`	the settler	the user (or anyone after the deadline) cancels the escrow	the user may cancel at any time on the contract; anyone after the escrow deadline, 2 hours after signing	"The swap did not start. Your 0.01 ETH is safe in the swap contract. [Get it back]"
A solver committed and did not pay the BTC	`refundable`	the settler, plus the solver's bond	anyone	after the Bitcoin deadline: 144 Bitcoin blocks after the commit, about a day	"The swap did not complete. Your 0.01 ETH is safe in the swap contract. [Get it back]"

What happened	`state`	Where the money is	Who acts	When	Show the user
Refund sent	`refunded`	the user's wallet: the escrow, and the solver's bond if there was one	nobody		"0.01 ETH is back in your wallet."
The order expired before anything happened (Solana)	`expired`	the user's wallet	nobody		"The order expired before it started. Nothing was sent."
Something else (Solana)	`failed`	the user's wallet, or the program's escrow until it expires	Maxy support		"Your coins are safe. Contact support with order 0xa46d…0c50."

The bond. When a solver commits and then does not pay, the escrow goes back to the user. The solver's bond (1% of the escrow by default) is split: half to whoever sends the refund transaction, half to the user. When the user sends it, the user gets all of it.

5.10c Getting the coins back

The refund is an on-chain call to the settler, sent from the user's wallet. Anyone may send it after the deadline, and the coins still go only to the user. `GET /orders/{orderId}/refund` builds it and simulates it from that wallet (with your key). Then the user's wallet sends the `transaction`. The samples use `BASE` from the setup.

curl

```
curl -s "$BASE/orders/$ORDER_ID/refund?chainId=8453"
```

TypeScript

```
const refund = await fetch(`${BASE}/orders/${orderId}/refund?chainId=8453`).then((r) =>
r.json());
if (refund.refundable) {
  const { from: _from, ...tx } = refund.transaction;
  const txHash = await walletClient.sendTransaction({ ...tx, value: 0n }); // the user's wallet;
it pays the gas
}
```

Python

```
refund = requests.get(f"{BASE}/orders/{order_id}/refund", params={"chainId": 8453}).json()
if refund["refundable"]:
    tx_hash = send_transaction(refund["transaction"])  # from the setup on buy BTC from an EVM
chain
```

A real sandbox answer, for an order in `refundable`:

Response


```
{
  "orderId": "0x53414e44424f58ec837ff3f2e3b5ff0db94324f8ff30f5e2d799de1f997273cd",
  "refundable": true,
  "function": "cancelBuyEscrow",
  "explanation": "No solver committed before the escrow deadline: the escrow goes back to the
user. Sandbox: nothing to send; the order reads refunded shortly.",
  "transaction": {
    "chainId": 8453,
    "from": "0x30A7cba29a6f3C34c812fc64586f61A5872251A4",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f58ab2ab7c49ce88c7bddd2bb3fc3d3900a32641cf1bd7a1372be",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": {
    "ok": true
  },
  "sandbox": true
}
```


`GET /orders/{id}/refund`: a real sandbox answer, 13 November 2026 (`x-request-id` a84971e0-80f7-4991-8f4d-01d03dcf56d9).

A real answer from the live API, for an order with no open escrow:

Response

```
{
  "orderId": "0xe34326e3978042566f4be2f97054e179c101e165b6c47872237111c246915ec0",
  "refundable": false,
  "reason": "There is no open escrow or unfilled commitment for this order; nothing to refund.",
  "unlocksAt": null,
  "transaction": null
}
```

Field	Meaning
`refundable`	`true` when the refund can be sent now
`function`	`cancelBuyEscrow` (no solver committed), `slashExpired` (a solver committed with a bond) or `refundExpired` (no bond)
`transaction`	`{ chainId, from, to, data, value }`: the settler call, simulated from `from` (the paying wallet, or `?from=` another sender)
`reason`, `unlocksAt`	When `refundable` is `false`: why, and for an escrow no solver committed to, when it opens (Unix seconds)

With the SDK:

TypeScript

```
// maxy and evm as in the quickstart; orderId is the order's id; account.address the user's
wallet
const plan = await maxy.swap.planRefund(8453, orderId, account.address); // which call applies,
and when it opens
const { txHash } = await maxy.swap.refund({ chainId: 8453, intentHash: orderId, evm }); // sent
from the user's wallet
```

Before the deadline `planRefund` answers `not_refundable`; for an escrow no solver committed to, `unlocksAtOf(error)` gives the time it opens. Pausing the settler never blocks these calls.

Without the SDK, call the same function on the settler with the order id, from the user's wallet.

5.10d Support script

1. Ask for the order id, or any transaction hash of the order, and look it up: `GET /status?txHash=…`.
2. Read `state`:
   - `completed`: send the user the `btcTxId` link (`https://mempool.space/tx/<btcTxId>`). A Bitcoin wallet shows the payment after its first confirmation, about 10 minutes.
   - `pending` or `in_progress` for under an hour: nothing is wrong; it is in flight.

- `refundable`: the user takes the coins back from the contract (above). Nothing is lost.
- `refunded`: point to the refund transaction in `txs`.
- Anything else, or a sell with a mismatched payment: send Maxy the order id, the transaction hashes and the time.

5.10e Errors on this page

Code	Cause	Fix
`not_refundable`	Before the deadline, or nothing to refund	Wait until `unlocksAtOf(error)`
`tx_reverted`	The refund call would revert (already refunded or settled)	Read the order's status
`user_rejected`	The user declined the refund transaction	Ask again

To build the refund screen, produce one on demand in the sandbox with `scenario=refund_available`.

Next: 5.11 When a sell goes wrong.

Last verified: 13 November 2026, against the sandbox.

5.11 When a sell goes wrong

After this page your support team can say what happens to a BTC payment that was short, over, late or never sent.

Before you start: 5.10 Refunds and failures. **Time:** 3 minutes.

5.11a Selling BTC: what can go wrong

The payout is locked before the user sends any BTC, so the risk is on the BTC the user sends.

What happened	`state`	What happens	Show the user
Paid the exact amount, in time	`in_progress` → `completed`	The proof releases the payout to the user	"Done. 38.01 USDC is in your wallet."
Paid the exact amount, confirmed one block after the deadline	`in_progress`	Still the user's: `swap.claimBtcToEvm` or `GET /orders/{id}/claim` proves it in that one-block window	"Your payment arrived late; claiming your payout."
Paid a different amount (less or more)	stays `awaiting_deposit`, then `expired`	The contract cannot prove a payment that is not exactly `order.deposit.sats`, so it cannot pay the user. Contact Maxy support with the Bitcoin txid	"This payment did not match the order. Contact support with your transaction id."
Paid more than one block after the deadline	`expired`	The escrow goes back to the solver; nothing on chain pays the user. Contact Maxy support with the txid	"This payment arrived after the order closed. Contact support with your transaction id."
Never paid	`expired`	The escrow goes back to the solver; nothing was sent	"The order expired. Nothing was sent."

Prevent the rows that need support. Show the exact amount, not a rounded one, and ask for one payment. Show the time left from `order.deposit.expiresAt` (`expiresAt` on the deposit route). Never let a wallet bump or split the payment.

The solver sends the proof itself. If a paid order sits unproven, `GET /orders/{orderId}/claim?chainId=8453` answers the proof as a simulated `transaction`. Any wallet may send it, and the payout still goes only to the user. It answers `{ "claimable": false, "reason", "retryable" }` until the payment has its confirmations.

5.11b Errors on this page

Code	HTTP	Cause	Fix
`quote_expired`	409	The deposit window passed	A new quote; do not pay
`deposit_address_unverified`		The payout is not escrowed for the user in the settler	**Do not pay.** Start again

Next: 5.12 Validate recipient addresses.

Last verified: 5 November 2026, against the SDK source and the API it runs.

5.12 Validate recipient addresses

After this page you check every address before you quote, and show the user why one cannot be paid.

Before you start: your key (a test key works). **Time:** 5 minutes.

5.12a The call

`GET /addresses/validate?chain=…&address=…` answers whether an address is well formed and whether a Maxy order can pay it. `chain` is `bitcoin`, an EVM chain (name, id or CAIP-2), or `solana`. The samples use `BASE` from the setup.

curl

```
curl -s "$BASE/addresses/validate?
chain=bitcoin&address=bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq"
```

TypeScript

```
const address = userInput.trim().replace(/^bitcoin:/i, '').split('?')[0]; // what the user
pasted or scanned
const check = await fetch(`${BASE}/addresses/validate?
chain=bitcoin&address=${encodeURIComponent(address)}`).then((r) => r.json());
if (!check.payable) showFieldError(check.reason); // your form's error line
```

Python

```
address = user_input.strip().removeprefix("bitcoin:").split("?")[0]  # what the user pasted or
scanned
check = requests.get(f"{BASE}/addresses/validate", params={"chain": "bitcoin", "address":
address}).json()
if not check["payable"]:
    print(check["reason"])
```

Response

```
{
  "address": "bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq",
  "chain": "bitcoin",
  "valid": true,
  "type": "p2wpkh",
  "network": "mainnet",
  "payable": true,
  "reason": null
}
```

a native SegWit address: a real answer from the live API, 2 November 2026 (`x-request-id` fd9bfdeb-8f74-484c-863f-41babfb19608).

Response


```
{
  "address": "bc1p5d7rjq7g6rdk2yhzks9smlaqtedr4dekq08ge8ztwac72sfr9rusxg3297",
  "chain": "bitcoin",
  "valid": true,
  "type": "p2tr",
  "network": "mainnet",
  "payable": false,
  "reason": "Receive BTC on a native SegWit address (bc1q...). Taproot and legacy addresses are
not paid by this route."
}
```


a Taproot address, valid but not paid by EVM buys: a real answer from the live API, 2 November 2026.

Response


```
{
  "address": "tb1qw508d6qejxtdg4y5r3zarvary0c5xw7kxpjzsx",
  "chain": "bitcoin",
  "valid": true,
  "type": "p2wpkh",
  "network": "testnet",
  "payable": false,
  "reason": "Not a Bitcoin mainnet address: Testnet and regtest addresses are not accepted; use
a Bitcoin mainnet address."
}
```


a testnet address: a real answer from the live API, 2 November 2026.

Field	Meaning
`valid`	The address is well formed for that chain and network
`type`	Bitcoin: `p2wpkh`, `p2wsh` (native SegWit, `bc1q…`), `p2tr` (Taproot, `bc1p…`), `p2pkh` (`1…`), `p2sh` (`3…`)
`network`	`mainnet`, `testnet`, `signet` or `regtest`
`payable`	A Maxy order would accept it as the destination
`reason`	Why not, in a sentence you can show; `null` when payable

5.12b Which addresses receive

Route	Receives at
EVM to BTC (all `evm-btc-*`)	Native SegWit only: `bc1q…` (P2WPKH or P2WSH)
SOL to BTC	Native SegWit and Taproot: `bc1q…`, `bc1p…`
BTC to EVM	The EVM wallet that signs, or `destAddress`

Maxy pays mainnet only, so a `tb1…` (testnet or signet) address is valid but never payable. The sandbox answers the same: its deposit addresses are `tb1q…`, and your recipient address must still be a mainnet one.

A malformed address answers `200`:

Response

```
{
  "address": "bc1qnotanaddress",
  "chain": "bitcoin",
  "valid": false,
  "type": null,
  "network": null,
  "payable": false,
  "reason": "Bitcoin address has an invalid character.",
  "sandbox": true
}
```

`GET /addresses/validate`: a real sandbox answer, 13 November 2026 (`x-request-id` cb3e068c-6874-41d2-a745-b1a112930c9d).

5.12c Paste and QR scan

- Strip a `bitcoin:` prefix and anything after `?` (a BIP-21 URI), and trim spaces, before validating.
- Validate on paste, not on every keystroke.
- Show the result next to the field, in words: the `reason` when it is not payable.
- Read the address back to the user in short form (`bc1qar0s…5mdq`) on the confirm screen.

5.12d Errors on this page

Code	HTTP	Cause	Fix
`invalid_input`	400	`chain` or `address` is missing, or `chain` is not one Maxy knows	Send both; `chain=bitcoin`, an EVM chain or `solana`

An address that is malformed is not an error: the answer is `200` with `valid: false` and a `reason`.

Next: 6.1 Showing a quote.

Last verified: 13 November 2026, against the sandbox.

6.1 Showing a quote

After this page your screen shows the amount, the time and the size limit the way users expect.

Before you start: 3.2 Pricing anatomy. **Time:** 5 minutes.

6.1a Showing a quote

- **Lead with what arrives.** "You get 0.00025282 BTC", from `formatted.receive`. Put what the user pays second.
- **One number.** `toAmount` is what reaches the user's Bitcoin address. Nothing is added or taken after signing, so there is no second total to show.
- **Prices while typing, a quote at confirm.** Call `GET /price` as the user types (debounce 300 ms). Get the quote when they press the button that leads to signing.
- **The countdown.** A quote holds for 60 seconds. Count down from `expiresAt` using the server's clock: read the `Date` response header and work from the offset, so a device clock five minutes off does not expire every quote. Get a new quote 5 seconds before expiry while the user is still reviewing; never let them sign an expired one.
- **Time.** Say "Your BTC is usually sent within seconds of your transaction confirming." Do not promise a number of seconds.

6.1b The size of an order

When the quote carries `issues.orderCap`, show its `message` as written, keep the amount visible, and disable the confirm button:

> Order size is capped for this API key. Try $100 or less.

Offer a one-tap fix ("Use $100") that fills in `issues.orderCap.capUsd`. Do not show this on the price while the user types; it belongs at the moment they move to order.

Below the smallest order: "Too small: orders start at 546 sats after costs. Try a larger amount."

6.1c Errors on this page

Code	HTTP	Cause	Words
`quote_expired`	409	The quote ran out at signing	"Prices moved. Here is a new quote."
`order_cap_exceeded`	422	Above your key's cap	The `message` as written
`amount_out_of_range`	422	Below the smallest order	"Too small: orders start at 546 sats after costs. Try a larger amount."

Next: 6.2 The signing moment.

Last verified: 5 November 2026, against the SDK source and the API it runs.

6.2 The signing moment

After this page your screen says what each signature does before the wallet opens.

Before you start: 6.1 Showing a quote. **Time:** 3 minutes.

6.2a What to say

This is where a user decides whether to trust your integration. Say what the signature does before the wallet opens:

Route	Before the wallet opens	What the wallet shows
ETH, BNB, AVAX (escrow)	"Sign your order, then confirm one transaction. Your ETH is held by the swap contract until your BTC is sent."	An EIP-712 message (the intent: amount, price, Bitcoin script, deadline), then a transaction to the settler
USDC (gasless)	"Sign in your wallet (once or twice). No gas. Your USDC is held by the swap contract until your BTC is sent."	A USDC permit (when Permit2's allowance is short) and a Permit2 transfer bound to this order
SOL	"Approve in your wallet. Your SOL is held by the swap program until your BTC is sent."	A readable quote request, then the escrow deposit

Show the receiving address in short form next to the button (`bc1qar0s…5mdq`) so the user checks it once more.

What each field binds is on security model. In the sandbox the wallet shows `MaxySandboxOrder`, which authorizes nothing (sandbox).

6.2b Errors on this page

Code	Cause	Words
`user_rejected`	The user declined in the wallet	Nothing. Return to the confirm screen
`wrong_chain`	The wallet could not switch chain	"Switch your wallet to Base to continue."

Next: 6.3 Recipient address entry.

Last verified: 5 November 2026, against the SDK source and the API it runs.

6.3 Recipient address entry

After this page your address field takes a paste or a scan, checks it, and tells the user where the BTC goes.

Before you start: 5.12 Validate recipient addresses. **Time:** 3 minutes.

6.3a Where the BTC goes

Say it plainly, once, near the address field:

> Your BTC goes straight to your own Bitcoin address. Maxy never holds it.

Address entry.

- Buying from an EVM chain, accept native SegWit addresses (`bc1q…`); from Solana, also Taproot (`bc1p…`).
- Check every address with `GET /addresses/validate?chain=bitcoin&address=…` before quoting, and show its `reason` when it is not payable.
- On paste or QR scan, strip a `bitcoin:` prefix and anything after `?`, and trim spaces.
- A testnet address (`tb1…`) on mainnet: "This is a testnet address. Use a mainnet address starting with bc1q."
- Prefill the user's own BTC address when your product has one, and still show it for confirmation.

6.3b Errors on this page

An address Maxy cannot pay is not an error: `GET /addresses/validate` answers `200` with `payable: false` and a `reason` to show. A quote with such an address answers `400 invalid_input`.

Next: 6.4 Order states and copy.

Last verified: 5 November 2026, against the SDK source and the API it runs.

6.4 Order states and copy

After this page every state of an order has its title and its line of text on your screen.

Before you start: 3.1 Order lifecycle. **Time:** 3 minutes.

6.4a The words

`state`	Title	Line under it
`pending`	Confirming your transaction	"Waiting for your transaction to confirm on Base."
`awaiting_deposit`	Send your BTC	"Send exactly 0.0005 BTC to the address below, in one payment." (selling)
`in_progress`	Sending your BTC	"Your BTC is on its way." (selling: "Payment seen. Waiting for 1 confirmation.")
`completed`	Sent	"0.00025282 BTC is on its way to your wallet. It shows there after the first confirmation, about 10 minutes." with the Bitcoin transaction link (selling: "Done. 38.01 USDC is in your wallet.")
`refundable`	Your coins can come back	"The swap did not complete. Your 0.01 ETH is safe in the swap contract. [Get it back]"
`refunded`	Refunded	"0.01 ETH is back in your wallet." with the refund transaction link
`cancelled`	Cancelled	"Nothing was sent."
`expired`	Expired	"The order expired before it started. Nothing was sent."
`failed`	Something went wrong	"Your coins are safe. Contact support with order 0xa46d...0c50."

A buy reads `completed` the moment `btcTxId` is set. Show success then, with the transaction link. Never show a settlement to the user: it is Maxy getting its own money back, and if it fails the swap still succeeded.

Links. Bitcoin transactions: `https://mempool.space/tx/<btcTxId>`. EVM transactions: the chain's explorer (`https://basescan.org/tx/<hash>` on Base).

6.4b Errors on this page

States are not errors. Refusals have their own words on failures and refunds.

Next: 6.5 Failures and refunds.

Last verified: 5 November 2026, against the SDK source and the API it runs.

6.5 Failures and refunds

After this page each refusal and each failed order shows words a user can act on.

Before you start: 6.4 Order states and copy. **Time:** 3 minutes.

6.5a What to say

Situation	Code	Words
The user rejected the signature	`user_rejected`	Nothing. Return to the confirm screen with the quote still valid
The quote expired at signing	`quote_expired`	"Prices moved. Here is a new quote." and show it
Not enough funds	`issues.balance`, `insufficient_funds`	"You need 0.0102 ETH on Base, including network costs."
Over the cap	`issues.orderCap`, `order_cap_exceeded`	The `message` as written
Rate limited	`rate_limited`	"Busy right now. Trying again..." and retry after `Retry-After`
Refund available	`refundable`	See the table above; the button calls `swap.refund` from the user's wallet

Never show a raw error code or stack to a user; log the `requestId`.

Support has one page to bookmark: refunds and failures.

6.5b Errors on this page

The table above is the list. Every code is in error codes.

Next: 6.6 Accessibility.

Last verified: 5 November 2026, against the SDK source and the API it runs.

6.6 Accessibility

After this page your swap screens meet the checks partners and their users ask about.

Before you start: nothing. **Time:** 2 minutes.

6.6a The rules

Meet WCAG 2.1 AA:

- State changes go to an `aria-live="polite"` region; the countdown does not announce every second (announce at 10 seconds and at expiry).
- Never carry meaning by colour alone: every state has its title in words.
- Contrast at least 4.5:1 for text; focus is visible on every control; the confirm button is reachable by keyboard.
- Amounts are text, not images, so screen readers and copy-paste work.

These are the rules Maxy asks of your screens. `@maxy/widget` follows them and is tested with axe at WCAG 2.1 AA in four browsers (`packages/maxy-widget/test/e2e.mjs`); on your own screens, conformance is yours to test.

6.6b Errors on this page

None: this page has no call.

Next: 6.7 Brand assets.

Last verified: 5 November 2026, against the SDK source and the API it runs.

6.7 Brand assets

After this page you can name Maxy and use its logo correctly.

Before you start: nothing. **Time:** 1 minute.

6.7a Naming Maxy

Write "Maxy" (one word, capital M). "Powered by Maxy" under the swap is welcome and optional.

6.7b The logo



Use the logo file as it is: do not recolour, stretch or redraw it.

6.7c Errors on this page

None: this page has no call.

Next: 7.1 Keys and secrets.

Last verified: 5 November 2026, against the SDK source and the API it runs.

7.1 Keys and secrets

After this page your security reviewer knows where each secret lives and what happens if one leaks.

Before you start: 3.6 Security model. **Time:** 3 minutes.

7.1a Where each secret lives

Secret	Where it lives	If it leaks
Your API key (`mxk_live_…`)	Your server's secret store	Ask Maxy to rotate it (keys). It cannot move funds or sign; someone holding it could spend your rate limit, read your order list and register webhooks
Your webhook signing secret (`whsec_…`)	Your server's secret store, next to the handler	Delete the webhook endpoint and create a new one; the new endpoint gets a new secret
Your users' wallet keys	Their wallets, or your own key management	Maxy never sees them; nothing in the API asks for one

Keys travel only in request headers, never in URLs. Every answer carries `x-request-id`; log it, not the key.

7.1b Errors on this page

Code	HTTP	Cause	Fix
`invalid_api_key`	401	A malformed or unknown key	Check the key
`key_revoked`	403	The key was revoked	Use your new key

Next: 7.2 Webhook security.

Last verified: 5 November 2026, against the SDK source and the API it runs.

7.2 Webhook security

After this page your handler accepts only deliveries from Maxy and processes each one once.

Before you start: 5.7 Track orders and receive webhooks. **Time:** 2 minutes.

7.2a The rules

Every delivery is signed with your endpoint's secret. Verify the signature on the raw body before you parse it, reject timestamps older than 5 minutes, and process each event `id` once. The header format, the verification code in Node and Python, and a test vector are on track orders and receive webhooks.

7.2b Errors on this page

What	Cause	Fix
Your verifier refuses a delivery	A wrong secret, a parsed body, or a clock more than 5 minutes off	Verify the raw bytes; check the secret and your clock

Next: 7.3 Content Security Policy.

Last verified: 5 November 2026, against the SDK source and the API it runs.

7.3 Content Security Policy

After this page your browser pages can reach Maxy under a strict policy.

Before you start: nothing. **Time:** 3 minutes.

7.3a The hosts

A browser page that calls Maxy directly needs the API host in `connect-src`:

HTTP

```
Content-Security-Policy: connect-src 'self' https://whhxyutitykcsdpvdgcd.supabase.co
```

A page that runs the TypeScript SDK in the browser also reads chains and Bitcoin data. Add the RPC endpoints you configure (`rpcUrls`), or the SDK's public defaults, and the Bitcoin explorers it uses:

HTTP

```
connect-src 'self' https://whhxyutitykcsdpvdgcd.supabase.co https://mainnet.base.org
https://base.llamarpc.com
  https://arb1.arbitrum.io https://bsc-dataseed.bnbchain.org https://bsc-rpc.publicnode.com
https://api.avax.network
  https://rpc.mainnet.chain.robinhood.com https://api.mainnet-beta.solana.com
https://mempool.space
  https://blockstream.info https://li.quest
  https://2sryoes37omcz5khnit2nuzhqy0xdayx.lambda-url.us-east-1.on.aws
  https://j77ctp5ohudgrddbciv2q2qwge0anxcy.lambda-url.us-east-1.on.aws
https://8mtf29qnke.execute-api.us-east-1.amazonaws.com
```

The last three hosts serve Solana routes; leave them out if you do not offer SOL.

The full list of hosts, per feature, is in endpoints. A page that only talks to your own server, which calls Maxy, needs none of these.

7.3b Errors on this page

A blocked host shows in the browser console as a CSP violation and in the SDK as `network_error`. Add the host named there.

Next: 7.4 Verifying contract addresses.

Last verified: 5 November 2026, against the SDK source and the API it runs.

7.4 Verifying contract addresses

After this page you can check that a transaction goes to the settler this guide names.

Before you start: 10.9 Contracts. **Time:** 3 minutes.

7.4a How to check

Every address a wallet signs for or pays is compiled into the SDK; a server response cannot add one. The settler on each chain, with links to its verified source, is on contracts. Before going live, compare the `to` of a real ETH escrow transaction from your tests with that table. A gasless escrow is sent by the solver's Safe, so its `to` is the Safe.

`GET /chains` returns each chain's settler, and it matched the table on contracts on the date below.

curl

```
curl -s "$BASE/chains" | jq '.chains[] | {name, settler}'
```

TypeScript

```
const { chains } = await fetch(`${BASE}/chains`).then((r) => r.json());
```

Python

```
chains = requests.get(f"{BASE}/chains").json()["chains"]
```

Response


```
{
  "chains": [
    {
      "chain": "bitcoin",
      "name": "Bitcoin",
      "type": "bitcoin",
      "nativeSymbol": "BTC",
      "settler": null,
      "explorer": "https://mempool.space",
      "caip2": "bip122:000000000019d6689c085ae165831e93"
    },
    {
      "chain": 8453,
      "name": "Base",
      "type": "evm",
      "nativeSymbol": "ETH",
      "settler": "0xeC5a05b28dd848F628e19CB26Df32f053e64ab86",
      "explorer": "https://basescan.org",
      "caip2": "eip155:8453"
    },
    "…"
  ],
  "sandbox": true
}
```


`GET /chains`: a real sandbox answer, 13 November 2026 (`x-request-id` a91d0dd7-3986-4030-b99b-522a86ea8d4a), with long fields folded.

7.4b Errors on this page

This call takes your key and has no error of its own beyond `401` without one.

Next: 7.5 What Maxy does and does not do.

Last verified: 13 November 2026, against the sandbox.

7.5 What Maxy does and does not do

After this page your reviewer has a plain statement of what Maxy holds, stores and checks.

Before you start: 3.6 Security model. **Time:** 2 minutes.

7.5a The statement

Maxy holds no user funds and performs no identity checks, so your product must do what the partner obligations list.

Maxy	
Holds user funds	No. Coins wait in the settler contract; the BTC goes from the solver to the user's address
Holds keys	No
Keeps user accounts or identity data	No identity data. Buying keeps no account. Selling signs the wallet in (Sign-In with Ethereum), which creates a wallet profile (address and username) on first use
Performs identity checks (KYC)	No
Screens recipient addresses	No
Stores	Per order: the signed intent, the signatures, the payout script, the status. Per call: the operation, time, IP address and API key, for rate limits and metering (data handling)

7.5b Errors on this page

None: this page has no call.

Next: 7.6 Partner obligations.

Last verified: 5 November 2026, against the SDK source and the API it runs.

7.6 Partner obligations

After this page you know what stays with you because Maxy keeps no accounts and checks no identities.

Before you start: 7.5 What Maxy does and does not do. **Time:** 2 minutes.

7.6a What stays with you

Maxy keeps no accounts and performs no identity checks. So the rules for your users apply in your product, as for any other trade there:

- Your onboarding and identity checks, where your licences require them.
- Address screening, where your policy requires it, before you quote.
- Your users' records: store `orderId`, both addresses, amounts and transaction hashes with your own order.
- Your terms with your users, including who they contact for support.

7.6b Errors on this page

None: this page has no call.

Next: 7.7 Screening and decline codes.

Last verified: 5 November 2026, against the SDK source and the API it runs.

7.7 Screening and decline codes

After this page you know whether Maxy screens an address or declines an order for a compliance reason.

Before you start: 7.6 Partner obligations. **Time:** 1 minute.

7.7a Today

- Maxy does not screen recipient addresses in the swap path.
- No order is declined for a screening or jurisdiction reason. If that changes, an order will answer `recipient_declined` or `jurisdiction_blocked`, on the order and in its webhook; neither is sent today, and the changelog will say first.
- `externalCustomerId` on `POST /orders` carries your own customer reference on the order, in `GET /orders` and in its export. Maxy stores it as given and reads it for nothing.
- `GET /addresses/validate` checks form and network only. It says nothing about who owns an address.

If your policy requires screening, run it before you quote.

7.7b Errors on this page

None: no screening code exists.

Next: 7.8 Reporting an incident.

Last verified: 5 November 2026, against the SDK source and the API it runs.

7.8 Reporting an incident

After this page you know where to send a security issue or a stuck order, and what to include.

Before you start: nothing. **Time:** 2 minutes.

7.8a What to send

- **A security issue** (a way to make the SDK or API sign, pay or reveal something it should not): report it through your onboarding channel with Maxy, with the component and version, steps to reproduce, impact, and any transaction hashes or order ids. Do not test against other users' funds. The policy is 27.4 Security policy.
- **An order that is stuck or wrong:** send the `orderId`, or any transaction hash of the order, and the call's `requestId` (support and escalation).

Report a security issue with `POST /security-reports`: `{ "summary", "details", "replyTo" }`, no key needed. A person reads every report. There is no published response time yet: your Maxy contact agrees it with you during

---

7.8b Errors on this page

None: this page has no call.

Next: 8.1 Sandbox.

Last verified: 5 November 2026, against the SDK source and the API it runs.

8.1 Sandbox

After this page you can run a whole integration with a test key: every route, every failure and real webhooks, with nothing real behind any of it. New to it? Start with the sandbox in five minutes.

Before you start: a test key from Maxy (`mxk_test_` and 32 letters and digits). **Time:** 5 minutes.

8.1a Switching to it

Change the key, in the HTTP API, the SDK and the CLI. The base URL, routes and JSON stay the same.

curl

```
export BASE=https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api
export MAXY_API_KEY=mxk_test_…
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/key"
```

TypeScript

```
import { createMaxyClient } from '@maxy-labs/sdk';

const maxy = createMaxyClient(); // reads MAXY_API_KEY
maxy.sandbox; // true with a test key, false with a live one
```

Python

```
import os, requests
BASE = "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api"
r = requests.get(f"{BASE}/key", headers={"x-maxy-key": os.environ["MAXY_API_KEY"]})
print(r.json()["sandbox"])  # True
```

Every answer to a test key carries `"sandbox": true` and the header `X-Maxy-Sandbox: true`, errors included. `maxy` shows `[SANDBOX]` in its banner, its prompt and on every run.

8.1b What is real and what is not

	In the sandbox
Prices and quote amounts	Real: the same reads as `GET /price`. They go out as Maxy's public client, never with your test key
Routes, JSON shapes, error codes, the 60-second quote	The same as live

	In the sandbox
Your key's order size cap	Real: an order over it is flagged and refused as live
Webhooks	Real: signed deliveries to your URL, with the same signature scheme and retries
Rate limit	Real, and lower: 120 calls a minute per key, and 50 sandbox orders open at a time
Signatures	Checked: they must come from the paying wallet. What is signed authorizes nothing (below)
Orders	Not real. They live in a sandbox table, walk their states on a timer and reach no solver, settler or chain
Balances and allowances	Not read. `issues` is empty unless a scenario sets it, and no wallet needs funds
Transactions, deposit addresses, txids	Not real, and marked (below)

8.1c Nothing to lose

A test key cannot reach the order path: only the Maxy API answers it, from the sandbox, and the SDK with one reaches nothing else. A mistake still costs nothing:

- **A signature authorizes nothing.** Every signature step signs `MaxySandboxOrder { orderId, step, notice }` under the domain `Maxy Sandbox` with verifying contract zero. It is not a permit, a Permit2 transfer or a settler order, so no contract can use it.
- **A `transaction` moves nothing.** It goes to the zero address with `value` `"0"`, a text marker as `data` and `simulated: true`. Its `gas` is `"1"`, below what any transaction needs, so a node refuses it. `chainId` is the real one, so your chain-switching code runs. A wallet that drops `gas`, estimates again and sends it anyway moves no coins and pays only that gas. Take `value` from the transaction, as you would live: never send another amount to it.
- **A Solana transaction cannot land.** It has no instructions and a blockhash no cluster knows.
- **A PSBT cannot confirm.** It spends an input that does not exist and pays only an `OP_RETURN`.
- **A deposit address is a `tb1q` address**, which a mainnet wallet refuses. Pay nothing: a sell moves on by itself.

Order ids, transaction hashes and Bitcoin txids all start with `53414e44424f58`, which is "SANDBOX" in hex, so they are easy to spot in logs. Keep one branch in your code:

TypeScript

```
const hash = transaction.simulated ? transaction.simulatedHash : await
wallet.sendTransaction(transaction);
```

8.1d Walk an order

`examples/integrate/sandbox.sh`, `sandbox.ts` and `sandbox.py` each take one order to `completed` with a throwaway wallet that holds nothing. The sandbox in five minutes shows each answer. With the SDK, `swap.createOrder` and `swap.execute` do all of it: the signer signs the sandbox message and `sendTransaction` is never called.

8.1e Webhooks

Register an endpoint as on track orders and webhooks. Deliveries are real and signed the same way. Each event carries `"sandbox": true` at the top and in `data`, with the `scenario`:

Response

```
{
  "id": "evt_…",
  "type": "order.updated",
  "createdAt": "2026-11-03T16:50:12.418Z",
  "sandbox": true,
  "data": {
    "orderId": "0x53414e44424f58…", "route": "evm-btc-escrow", "chainId": 8453,
    "state": "completed", "previousState": "in_progress", "rawStatus": "SANDBOX_COMPLETED",
    "isComplete": true, "btcTxId": "53414e44424f58…", "txs": [ … ], "sandbox": true, "scenario":
"success"
  }
}
```

8.1f Errors on this page

Code	HTTP	Cause	Fix
`sandbox_key_refused`	403	A test key sent to another Maxy function	Test keys work only on the HTTP API
`not_found`	404	A real order id read with a test key	A test key sees only sandbox orders

Code	HTTP	Cause	Fix
`rate_limited`	429	Over 120 calls a minute, or 50 open orders	Wait `Retry-After` seconds, or let orders finish

---

8.1g What the sandbox does not do

- **It proves your integration, not your wallet or the chain.** Nothing is broadcast, so gas, nonces and your wallet's real transactions go untested. Finish with a few small mainnet orders.
- **A signature must be a plain wallet's.** Live, a contract wallet's signature is checked with ERC-1271. The sandbox reads no chain, so it recovers an ordinary signature only and otherwise answers `wallet_mismatch`.
- **Swaps only.** Loans, mints and wallet sign-in are not in it. With a test key those SDK calls, and the low-level order builders such as `swap.createGaslessOrder`, refuse with `sandbox_refused`, and other Maxy functions refuse a test key with `401 invalid_api_key` or `403 sandbox_key_refused`.
- **A test key sees only sandbox orders.** A real order id answers `404`. A sandbox id read with a live key answers from the sandbox and says so; without a key, `401`, as on every route.
- **Prices depend on the live quote service.** When it is down, sandbox prices fail as live ones do; `GET /health` says.

Next: 8.2 Sandbox scenarios.

Last verified: 5 November 2026, against the SDK source and the API it runs.

8.2 Sandbox scenarios

After this page you can force any outcome with a test key, and move an order on without waiting.

Before you start: a test key and the sandbox. **Time:** 5 minutes.

8.2a Timing

The clock starts at `POST /orders/submit`, or at `POST /orders` for a gasless order or a sell. An order nobody submits starts by itself after 30 seconds, as the solver would find an escrow on chain.

Route	States, in seconds from the start
Buying BTC, EVM to EVM, SOL to BTC	`pending`; `in_progress` at 8; `completed` at 20
Selling BTC, BTC to SOL	`pending`; `awaiting_deposit` at 5; `in_progress` at 15; `completed` at 27

Webhooks follow within about 15 seconds of each change, one event per state, in order. Sandbox orders are kept for 7 days.

8.2b Scenarios

Force an outcome with `scenario`: a query parameter on `GET /price` and `GET /quote`, a field in the body of `POST /orders`, or `scenario` on `swap.quote`, `swap.createOrder` and `swap.execute`. A scenario named on the quote stays with the order, and a live key ignores it. `GET /sandbox/scenarios` lists them.

`simulate` works as `scenario` does, and also takes the names `quote_expired`, `fill_timeout`, `underpaid` and `refund`. To skip the waiting, `POST /sandbox/orders/{id}/advance` moves an order to its next state now, or to `{ "to": "completed" }`: see move an order on.

`scenario`	What happens
`success`	The default: the order reaches `completed`
`refund_available`	The order reaches `refundable`. `GET /orders/{id}/refund` then answers `refundable: true` with a sandbox transaction, and the order reads `refunded` 15 seconds after you ask
`order_expired`, `order_failed`, `order_cancelled`	The order walks to `expired`, `failed` or `cancelled`
`expired_quote`	`POST /orders` answers `409 quote_expired`. Waiting 61 seconds does the same

`scenario`	What happens
`over_cap`	The quote shows the amounts with `issues.orderCap` and `steps: null`; `POST /orders` answers `422 order_cap_exceeded`. A quote above your test key's cap does the same
`too_small`	`GET /price` and `GET /quote` answer `422 amount_out_of_range`
`unfillable`	The quote shows the amounts with `liquidityAvailable: false` and no steps; `POST /orders` answers `422 insufficient_liquidity`. No figure is named, as live
`price_moved`	`POST /orders` answers `409 quote_mismatch`
`rate_limited`	The call answers `429 rate_limited` with `Retry-After: 5`
`insufficient_funds`	The quote sets `issues.balance`; on the gasless route `steps` is null
`needs_approval`	The quote sets `issues.allowance` and adds an `approve` step with a sandbox transaction
`simulation_fails`	`POST /orders` answers `simulation.ok: false` with a reason
`underpaid`	A sell whose payment came in short: `awaiting_deposit`, then `expired` with `rawStatus` `SANDBOX_UNDERPAID`; `txs` lists the `btc_payment` and the `btc_return` that sends it back to the payer, as live
`webhook_replay`	The order completes, and each webhook event arrives twice with the same `id`

curl

```
curl -sG -H "x-maxy-key: $MAXY_API_KEY" "$BASE/quote" \
  --data-urlencode "from=ETH on base" -d to=BTC -d amount=0.01 \
  -d fromAddress="$WALLET" -d toAddress="$MY_BTC_ADDRESS" -d scenario=refund_available
```

TypeScript

```
const { status } = await maxy.swap.execute({
  from: 'ETH on base', to: 'BTC', amount: '0.01',
  evm, btcAddress, scenario: 'refund_available',
});
// status.state === 'refundable'
const { txHash } = await maxy.swap.refund({ chainId: 8453, intentHash: status.orderId as
`0x${string}`, evm });
```

Python

```
quote = requests.get(f"{BASE}/quote", headers=HEADERS, params={
    "from": "ETH on base", "to": "BTC", "amount": "0.01",
    "fromAddress": wallet.address, "toAddress": btc_address, "scenario": "refund_available",
}).json()
```

Response

```
{
  "scenarios": [
    {
      "name": "success",
      "what": "The default. The order walks pending, in_progress, completed, with a sandbox
btcTxId."
    },
    {
      "name": "refund_available",
      "what": "The order reaches refundable. GET /orders/{id}/refund then answers a refund; the
order reads refunded 15 s after you ask."
    },
    {
      "name": "order_expired",
      "what": "The order walks pending, expired: an escrow that was never sent."
    },
    "…"
  ],
  "sandbox": true
}
```

`GET /sandbox/scenarios`: a real sandbox answer, 13 November 2026 (`x-request-id` c628ef67-3029-4b07-b3fe-78be13aca7b4), with long fields folded.

8.2c Move an order on

Moves a sandbox order to its next state, or a later one on its walk. A state its scenario does not reach answers `409 order_state`, with `allowedStates`.

curl

```
curl -s -X POST "$BASE/sandbox/orders/$ORDER_ID/advance" -H "x-maxy-key: $MAXY_API_KEY" -H
"content-type: application/json" -d '{"to":"completed"}'
```

TypeScript

```
await maxy.sandboxControls.advance(orderId, { to: 'completed' });
```

Python

```
maxy.sandbox_controls.advance(order_id, to="completed")
```

Response


```
{
  "orderId": "0x53414e44424f5823936eadc65c68429f1fc465d671b4382365c5ff83ffbf13ae",
  "route": "evm-btc-escrow",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7",
  "txs": [
    {
      "label": "btc_payout",
      "hash": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "advanced": true
}
```


`POST /sandbox/orders/{id}/advance`: a real sandbox answer, 13 November 2026 (`x-request-id` 172f8ed3-4ac3-4e7f-a061-e290c75044a6).

Response


```
{
  "error": {
    "code": "order_state",
    "message": "This order's walk (scenario success) does not reach pending from completed. Next states: none. Pick the scenario at quote time.",
    "details": {
      "state": "completed",
      "allowedStates": []
    },
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#order_state"
  },
  "requestId": "1435fe63-cf16-48e3-b196-33b6bce8e92c",
  "sandbox": true
}
```


`POST /sandbox/orders/{id}/advance` (HTTP 409): a real sandbox answer, 13 November 2026 (`x-request-id` 1435fe63-cf16-48e3-b196-33b6bce8e92c).

`maxy sandbox advance <id> --to completed` does the same from a terminal. `GET /health?scenario=degraded` answers a degraded health in the same shape, so a router can test skipping Maxy.

8.2d Errors on this page

Code	HTTP	Cause	Fix
`invalid_input`	400	A scenario name that does not exist	`GET /sandbox/scenarios` lists them
`order_state`	409	A state this order's scenario does not reach	Read `allowedStates`

Next: 8.3 Test scenarios.

Last verified: 13 November 2026, against the sandbox.

8.3 Test scenarios

After this page you have run every scenario your support team will meet, first in the sandbox and then with a few small mainnet orders.

Before you start: a working quickstart, your path and a test key. **Time:** about an hour; the mainnet part costs a few dollars.

8.3a Test in the sandbox

Set `MAXY_API_KEY` to your test key. 8.1 Sandbox says what is and is not real.

Scenario	How to produce it in the sandbox	What you should see
Happy path, gasless	Quote `USDC on base` to BTC	`pending` → `in_progress` → `completed` with a `btcTxId` starting `53414e44424f58`; one `order.updated` webhook per change
Happy path, escrow	Quote `ETH on base` to BTC; `POST /orders/submit` with `transaction.simulatedHash` instead of sending	The same
Expired quote	`scenario=expired_quote`, or wait 61 seconds	`quote_expired` (HTTP 409) at `POST /orders`
Over the cap	`scenario=over_cap`, or quote above your test key's cap	`issues.orderCap` with the message, `steps: null`; `POST /orders` answers `422 order_cap_exceeded`
Too small	`scenario=too_small`	`422 amount_out_of_range`
Unfillable size	`scenario=unfillable`	The amounts with `liquidityAvailable: false` and no steps; `POST /orders` answers `422 insufficient_liquidity`
Price moved	`scenario=price_moved`	`409 quote_mismatch` at `POST /orders`
Not enough funds	`scenario=insufficient_funds`	`issues.balance` set; on the gasless route `steps` is null
Allowance short	`scenario=needs_approval`	`issues.allowance` set and an `approve` step
Failed simulation	`scenario=simulation_fails`	`simulation.ok: false` with a reason
User rejects	Reject the sandbox signature in the wallet	SDK: `user_rejected`; HTTP: you never call `POST /orders`. Nothing signed, nothing sent
Bad address	Validate `tb1q…` or a `bc1p…` for an EVM buy	`GET /addresses/validate` answers `payable: false` with a reason

Scenario	How to produce it in the sandbox	What you should see
Underpayment	`scenario=underpaid` on a sell	`awaiting_deposit`, then `expired` with `rawStatus` `SANDBOX_UNDERPAID`
Fill timeout	`simulate=fill_timeout`	`in_progress`, then `refundable`
Any state now	`POST /sandbox/orders/{id}/advance`	The order reads that state; webhooks follow
Maxy degraded	`GET /health?scenario=degraded`	`degraded: true`: your router skips the source
Track by hash	`GET /status?txHash=` with `simulatedHash`, then with the sandbox `btcTxId`	The same order both times
Webhook test	`POST /webhooks/{id}/test`	A signed test event, marked `sandbox`, that your handler verifies and records once
Webhook replay	`scenario=webhook_replay`	Each event arrives twice with the same `id`; your handler ignores the second
Refund screen	`scenario=refund_available`, then wait for `refundable`	`GET /orders/{id}/refund` answers `refundable: true`; the order reads `refunded` 15 seconds later. Your copy and button match refunds
Other endings	`scenario=order_expired`, `order_failed`, `order_cancelled`	The order walks to `expired`, `failed`, `cancelled`
Rate limit	`scenario=rate_limited`	`429` with `Retry-After`: you wait and retry

Every scenario is one extra parameter on the quote:

curl

```
curl -s -G "$BASE/quote" -H "x-maxy-key: $MAXY_API_KEY" --data-urlencode "from=ETH on base" -d
to=BTC -d amount=0.004 \
  -d fromAddress="$WALLET" -d toAddress="$MY_BTC_ADDRESS" -d scenario=refund_available
```

TypeScript

```
const q = new URLSearchParams({ from: 'ETH on base', to: 'BTC', amount: '0.004', fromAddress:
account.address, toAddress: btcAddress, scenario: 'refund_available' });
const quote = await fetch(`${BASE}/quote?${q}`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
quote = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, params={
    "from": "ETH on base", "to": "BTC", "amount": "0.004",
    "fromAddress": account.address, "toAddress": btc_address, "scenario": "refund_available",
}).json()
```

Response


```
{
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "4000000000000000",
  "toAmount": "10887",
  "toAmountMin": "10887",
  "formatted": {
    "send": "0.004 ETH on Base",
    "receive": "0.00010887 BTC",
    "pair": "ETH on Base → BTC"
  },
  "expiresAt": 1794535613,
  "expiresAtIso": "2026-11-13T02:06:53.000Z",
  "limits": {
    "minSats": 546
  },
  "quoteId": "0x53414e44424f58ec837ff3f2e3b5ff0db94324f8ff30f5e2d799de1f997273cd",
  "order": "…",
  "issues": {
    "balance": null,
    "allowance": null,
    "orderCap": null
  },
  "steps": [
    {
      "id": "intent",
      "kind": "signature",
      "description": "Sign the order (EIP-712). No gas. Sandbox: this signature authorizes
nothing.",
      "signatureKind": "eip712",
      "typedData": "…",
      "hash": "0x55bee581948655ac2e7fb482ada996695b657f7aef4b84ed1e9626aa0d9ba3d8"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signature; you get back the transaction to send.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    },
    {
      "id": "escrow",
      "kind": "transaction",
      "description": "Sandbox: do not send the transaction. POST /orders/submit with its
```

```
simulatedHash.",
      "chainId": 8453,
      "transaction": null
    }
  ],
  "signing": "…",
  "sandbox": true
}
```

`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 61d6f036-a9e2-4edc-b4a2-11e539be0cf2), with long fields folded.

8.3b Test small on mainnet

Only a real order proves your wallet, chain settings and key's cap. The cap keeps tests small; the smallest buy is 546 sats (well under $1).

Scenario	How to produce it on mainnet	What you should see
Happy path, gasless	Buy BTC with $5 of USDC on Base	`pending` → `in_progress` → `completed` with `btcTxId`; one `order.updated` webhook per change
Happy path, escrow	Buy BTC with 0.002 ETH on Base	The same, after your wallet sends the escrow transaction
Over the cap	Quote $150 with a $100-capped key	`issues.orderCap` with the message, `steps: null`; `POST /orders` answers `422 order_cap_exceeded`
Track by hash	`GET /status?txHash=` with your escrow transaction hash, then with the Bitcoin txid	The same order both times

Refunds, rate limits and solver failures cannot be produced on mainnet: test them in the sandbox. Do not load-test production.

8.3c Errors on this page

Each scenario above names the code it produces. All of them are in error codes.

Next: 8.4 Load and rate-limit behaviour.

Last verified: 13 November 2026, against the sandbox.

8.4 Load and rate-limit behaviour

After this page you know what a 429 looks like, how the SDK surfaces it, and how much load to test.

Before you start: 3.8 Rate limits, retries and idempotency. **Time:** 3 minutes.

8.4a What a 429 looks like

Response

```
{
  "error": {
    "code": "rate_limited",
    "message": "Too many requests (sandbox scenario rate_limited). Try again in 5 s.",
    "retryable": true,
    "docs": "https://maxy.co/docs/sdk-errors#rate_limited"
  },
  "requestId": "275a5744-a161-41e8-9bd6-8dc8327834a3",
  "sandbox": true
}
```

`GET /quote` (HTTP 429): a real sandbox answer, 13 November 2026 (`x-request-id` 275a5744-a161-41e8-9bd6-8dc8327834a3).

The answer carries `Retry-After`, and the API does not act on the call. Force it in the sandbox with `scenario=rate_limited`.

8.4b How the SDK handles it

`@maxy-labs/sdk/core` retries twice by itself, honouring `Retry-After`, and queues a burst when the last answer said the minute is used up (the SDK client). After that it raises `RateLimitError` with `retryAfterMs`. `swap.*` on the full client does not retry: it raises a `MaxyError` with code `rate_limited`, and `retryAfterSecondsOf(error)` gives the wait:

curl

```
curl -s -D - -o /dev/null "$BASE/price?from=USDC%20on%20base&to=BTC&amount=10" | grep -i -E
'ratelimit|retry-after'
```

TypeScript

```
import { isMaxyError, retryAfterSecondsOf } from '@maxy-labs/sdk';

try {
  return await maxy.swap.price(request);
} catch (error) {
  if (!isMaxyError(error, 'rate_limited')) throw error;
  await new Promise((r) => setTimeout(r, (retryAfterSecondsOf(error) ?? 1) * 1000));
  return maxy.swap.price(request);
}
```

Python

```
r = requests.get(f"{BASE}/price", params=params)
if r.status_code == 429:
    time.sleep(int(r.headers["retry-after"]))
    r = requests.get(f"{BASE}/price", params=params)
```

8.4c How much to test

Where	Limit	What to do
Sandbox	120 calls a minute per key, 50 open orders	Test your backoff here with the scenario above
Live	600 calls a minute per key (every route takes one)	Do not load-test production: quotes reach the solver that fills real orders

No published load figures exist. Ask Maxy before a launch that needs more (raising limits).

8.4d Errors on this page

Code	HTTP	Cause	Fix
`rate_limited`	429	Over the limit, or 50 sandbox orders open	Wait `Retry-After` seconds, or let orders finish

Next: 8.5 Go-live checklist.

Last verified: 13 November 2026, against the sandbox.

8.5 Go-live checklist

After this page you know what must be true before your first user swaps.

Before you start: the testing scenarios run. **Time:** 15 minutes.

8.5a The checklist

- [ ] Your `mxk_live_` key is in your server's secret store, not in a browser bundle or a repository.
- [ ] Your key's order size cap fits your users; ask Maxy to raise it if not.
- [ ] The cap message shows at the confirm step, as a message, with the amount still visible.
- [ ] Every BTC address is checked with `GET /addresses/validate` before quoting.
- [ ] Your code branches on `simulated` or on the key, so it can never send a sandbox `transaction`. No test key is in production.
- [ ] Every sandbox scenario above has run, and one small real swap each way has completed, with its `orderId` in your ledger.
- [ ] Your webhook endpoint is registered, verifies the signature, and processes each event `id` once.
- [ ] A publishable key in your page lists your production origins; the secret key is on your server only.
- [ ] Your ops alerting fires on `refundable`, and your users see the refund button.
- [ ] Every state and failure shows the words from UX guidelines.
- [ ] Browser pages: the API host is in your Content Security Policy.
- [ ] Your support team has the refunds page and your Maxy contact.

Each path adds its own lines: aggregator, embed, frontend, backend.

8.5b Errors on this page

None: this page has no call.

Next: 8.6 Launch review.

Last verified: 5 November 2026, against the SDK source and the API it runs.

8.6 Launch review

After this page you know what to send Maxy before launch and what to ask for.

Before you start: the go-live checklist. **Time:** 5 minutes.

8.6a What to send

Send your Maxy contact the `orderId`s of your test orders, your webhook URL and your key's prefix (`mxk_live_` and the next six characters). Ask then for the order size cap and rate limit your launch needs.

There is no scheduled review meeting or form: your Maxy contact reads what you send.

8.6b Errors on this page

None: this page has no call.

Next: 9.1 Status and maintenance.

Last verified: 5 November 2026, against the SDK source and the API it runs.

9.1 Status and maintenance

After this page you know how to check Maxy is up, and what to expect around maintenance.

Before you start: nothing. **Time:** 2 minutes.

9.1a Is Maxy up?

`GET /health` needs no key and answers within a second:

curl

```
curl -s "$BASE/health"
```

TypeScript

```
const health = await fetch(`${BASE}/health`).then((r) => r.json());
```

Python

```
health = requests.get(f"{BASE}/health").json()
```

Response


```
{
  "status": "ok",
  "degraded": false,
  "checkedAt": "2026-11-13T02:04:27.415Z",
  "components": {
    "api": {
      "status": "ok"
    },
    "quotes": {
      "status": "ok",
      "latencyMs": 158
    },
    "solver": {
      "status": "ok",
      "latencyMs": 158,
      "priceAgeSeconds": 8
    },
    "bitcoin": {
      "status": "ok",
      "tipHeight": 975617
    },
    "verifiers": [
      {
        "chainId": 8453,
        "chain": "Base",
        "status": "ok",
        "height": 975617,
        "lagBlocks": 0
      },
      {
        "chainId": 56,
        "chain": "BNB Chain",
        "status": "ok",
        "height": 975617,
        "lagBlocks": 0
      },
      {
        "chainId": 42161,
        "chain": "Arbitrum",
        "status": "ok",
        "height": 975617,
        "lagBlocks": 0
      },
      {
        "chainId": 43114,
        "chain": "Avalanche",
        "status": "ok",
```

```
        "height": 975617,
        "lagBlocks": 0
      },
      {
        "chainId": 4663,
        "chain": "Robinhood",
        "status": "ok",
        "height": 975617,
        "lagBlocks": 0
      }
    ]
  }
}
```


`GET /health`: a real sandbox answer, 13 November 2026 (`x-request-id` f37adee8-d4df-48d6-a6e2-2023eeecac80).

`degraded` is true when the solver does not answer, Bitcoin's tip cannot be read, or a chain's verifier is more than 3 blocks behind it. A router can call it before ranking Maxy and skip the source while it reads `true`, instead of timing out on a quote. `verifiers` lists each chain with its height and lag. The answer is shared per server for 30 seconds. In the sandbox it is simulated, and `?scenario=degraded` forces the flag.

Forced in the sandbox, the same shape, with `?scenario=degraded`:

Response


```
{
  "status": "degraded",
  "degraded": true,
  "checkedAt": "2026-11-13T02:09:21.000Z",
  "components": {
    "api": {
      "status": "ok"
    },
    "quotes": {
      "status": "degraded",
      "latencyMs": 120
    },
    "solver": {
      "status": "degraded",
      "latencyMs": 120,
      "priceAgeSeconds": 2
    },
    "bitcoin": {
      "status": "ok",
      "tipHeight": 900000
    },
    "verifiers": [
      {
        "chainId": 8453,
        "chain": "Base",
        "status": "degraded",
        "height": 899988,
        "lagBlocks": 12
      },
      "…"
    ]
  },
  "sandbox": true
}
```


`GET /health`: a real sandbox answer, 13 November 2026 (`x-request-id` dd7986b5-4259-40c5-bff9-cd2df10804f9), with long fields folded.

9.1b Incidents and maintenance

maxy.co/status shows the API, quotes, the solver, Bitcoin and each chain's verifier, read live, with any incidents. `GET /system/history` has each component per day, and `GET /system/maintenance` the calendar. Subscribe by signed webhook or RSS (`POST /system/subscribe` , `GET`

`/system/feed.xml`). An incident opens by itself after three failed checks in a row. No response time is promised yet. Your Maxy contact tells you about planned work and incidents in your shared channel, and agrees response times with you during onboarding.

9.1c Errors on this page

`GET /health` always answers `200`: its `degraded` field says when something is wrong.

Next: 9.3 Support and escalation.

Last verified: 13 November 2026, against the sandbox.

9.2 Your account

After this page you can manage your keys, read your usage and statements, look up an order and see whether Maxy is healthy, with your secret key and the API.

Before you start: a key. **Time:** 5 minutes.

9.2a Your keys and usage

Route	Does
`POST /sandbox/keys`	A test key and a test publishable key, no sign-up
`GET /account`	Your keys by prefix, calls per day, what you asked for
`POST /keys`	A publishable key for your origins
`POST /keys/rotate`	A new secret key with the same cap
`DELETE /keys/{prefix}`	Revoke a key
`POST /keys/{prefix}/origins`	Change a publishable key's origins
`POST /limits/requests`	Ask for a higher order size cap
`GET /explorer/{query}`	An order by id, intent hash, transaction or Bitcoin txid

---

9.2b Statements

curl

```
curl -s "$BASE/statements?month=2026-11" -H "x-maxy-key: $MAXY_API_KEY"
```

TypeScript

```
import { statements } from '@maxy-labs/sdk/server';
const pdf = await statements.download({ apiKey: process.env.MAXY_API_KEY, month: '2026-11',
format: 'pdf' });
```

Python

```
pdf = maxy.statements.download("2026-11", "pdf")
```

JSON, CSV or PDF: each order, and totals by state and route, for your keys in one environment. It is a record of orders, not an invoice. There are no fee lines.

9.2c Status

maxy.co/status reads `GET /health` live. A check fails only when a second probe agrees (the solver asked twice, each chain's verifier read from every RPC on its own), and an incident opens by itself only after checks have failed for 10 minutes without a pass, with a first note at once. It closes after two good checks. Subscribe by signed webhook with `POST /system/subscribe`, or read `GET /system/feed.xml`.

Every night the server walks one order through each branch on the sandbox, and `GET /system/verified` says how it went. The docs show that date.

9.2d Errors on this page

Code	Cause	Fix
`invalid_api_key`	A key Maxy did not issue, or revoked	Check it, or get a new one
`scope_denied`	A publishable key on an account route	Use your secret key

Next: 9.3 Support and escalation.

Last verified: 5 November 2026, against the SDK source and the API it runs.

9.3 Support and escalation

After this page you know who to write to about an order and what to include.

Before you start: nothing. **Time:** 2 minutes.

9.3a When an order needs help

There is no published SLA, maintenance calendar or escalation list: your Maxy contact agrees response times and an escalation path with you during onboarding. Write to them in your shared channel with:

1. The `orderId`, or any transaction hash of the order (the escrow transaction or the Bitcoin txid).
2. The `requestId` (or `x-request-id` header) of the call that failed, if one did.
3. When it happened, in UTC, and your key's prefix (`mxk_live_` and the next six characters).

`GET /status?txHash=…` shows you the same state Maxy sees. An order in `refundable` needs nothing from Maxy: the user takes the coins back from the contract (refunds).

9.3b Errors on this page

None: this page has no call.

Next: 9.4 Raising limits.

Last verified: 5 November 2026, against the SDK source and the API it runs.

9.4 Raising limits

After this page you know how to see your limits and how to ask for more.

Before you start: nothing. **Time:** 1 minute.

9.4a Ask your contact

Ask for a higher order size cap with `POST /limits/requests`. A person at Maxy answers, and `GET /limits/requests` shows the answer. `GET /key` shows your cap; the `RateLimit-*` headers show your rate limit.

The request raises nothing by itself: your cap stays as it is until Maxy answers.

9.4b Errors on this page

None: this page has no call. `GET /limits` is on order size and limits.

Next: 9.5 Staying current.

Last verified: 5 November 2026, against the SDK source and the API it runs.

9.5 Staying current

After this page you know where changes are announced and how to follow them.

Before you start: nothing. **Time:** 1 minute.

9.5a Where changes appear

- The changelog lists every change to the API and the SDK.
- The API only gains fields and routes; nothing is removed or renamed. A breaking change is announced to every key holder at least 90 days ahead, and the old behaviour keeps working for at least six months.
- The SDK files live at https://maxy.co/sdk/. `maxy-labs-sdk-latest.tgz` (npm) and `maxy-latest.tar.gz` (pip) are always the newest, and `SHA256SUMS` lists every file's SHA-256. To upgrade, install the same URL again. The changelog says what changed.
- The changelog is also an RSS feed: `https://maxy.co/docs/changelog.xml`.
- The deprecation policy states the 90-day window.

9.5b Errors on this page

None: this page has no call.

Next: 9.6 Troubleshooting.

Last verified: 5 November 2026, against the SDK source and the API it runs.

9.6 Troubleshooting

After this page you can find out why a call fails, starting from one command.

Before you start: the kit's SDK installed. **Time:** 2 minutes.

9.6a Start with maxy doctor

`npx maxy doctor` checks your key, its class and environment, your clock against Maxy's, the connection, and the solver, Bitcoin and each chain's verifier. With a publishable key, `--origin https://app.example` checks that page's origin. Every failure says what to do. It exits 1 if anything failed, and `--json` prints the same for a script.

```
◆ maxy  maxy doctor

✓ SDK: @maxy-labs/sdk 0.15.0, Node 22.11.0
✓ Key: secret, sandbox: nothing real can move (from MAXY_API_KEY)
✓ Connection: Maxy answered in 665 ms (request c41261c5-2d98-49c3-8a71-4dc0fc062f6d)
✓ Clock: Within 1 s of Maxy
✓ Maxy: Solver, Bitcoin and every verifier ok
✓ Key at Maxy: Known: mxk_test_Rj948T…, order size cap $100, 120 calls a minute
```

It says	It means	Fix
Not a whole key	A cut-off paste	A key is `mxk_live_`, `mxk_test_`, `mxk_pk_live_` or `mxk_pk_test_` and 32 letters and digits
A publishable key … not from this terminal	Wrong key class here	In a terminal or on a server, use the secret key
Maxy did not issue this key, or it was revoked	`invalid_api_key`, `key_revoked`	Ask Maxy for a new one
This machine is N s behind Maxy	A wrong clock	Turn on network time. Webhook verification allows 5 minutes
Degraded	`/health` reads degraded	Nothing on your side: skip Maxy as a source until it clears

---

9.6b Common symptoms

Symptom	Likely cause	Fix
`origin_not_allowed`	A publishable key from a page not on its list	Ask Maxy to add the origin (keys)
`scope_denied`	A publishable key placing an order	Send it from your server with the secret key
`quote_expired`	The user took longer than the quote's 60 seconds	Get a new quote and sign that; `quotes.refresh` asks the same question
`verification_failed` from `verifyWebhook`	The body was parsed before the check, or the clock is off	Pass the raw bytes; read `check` on the error
An order stays `pending`	An escrow transaction that never landed	Read its hash on the chain's explorer; `POST /orders/{id}/cancel` takes back an open escrow
Quotes answer `liquidityAvailable: false`	Larger than Maxy can fill now	Try a smaller amount

Quote `requestId` from the error to your Maxy contact (support).

9.6c Errors on this page

None: this page has no call of its own. `maxy doctor` reports what it finds in words.

Next: 10.1 HTTP API.

Last verified: 5 November 2026, against the SDK source and the API it runs.

10.1 HTTP API

This page is for integrators working in any language: price, quote and place native Bitcoin swaps with plain HTTP calls, no SDK needed. It is the Maxy SDK running on Maxy's side, so every answer carries the checks the SDK makes in a browser, and your users' wallets still sign and send everything themselves.

Base URL: `https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api` (`BASE` in the samples: `export BASE=https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api`) **Auth:** every route takes your Maxy API key in `x-maxy-key` (or `Authorization: Bearer mxk_live_…`), a key Maxy issued and has not revoked. Only `/`, `/openapi.json`, `/health`, the status page (`/system/*`) and `POST /sandbox/keys` (which gives you a test key) answer without one. A page uses a publishable key instead: reads, quotes and the user's own signed order, from its own origins (keys). **Sandbox:** the same base URL with a test key (`mxk_test_…`): same routes, same shapes, nothing real (sandbox). **Spec:** `/openapi.json` (OpenAPI 3.1), ready for Postman, Insomnia or a code generator.

10.1a A price in one line

Paste it into a terminal with your key (no signup for a test key: `POST /sandbox/keys`):

curl

```
curl "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/price?
from=ETH%20on%20base&to=BTC&amount=0.01" -H "x-maxy-key: $MAXY_API_KEY"
```

JavaScript

```
const res = await fetch('https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/price?
from=ETH%20on%20base&to=BTC&amount=0.01', { headers: { 'x-maxy-key': process.env.MAXY_API_KEY }
});
const { formatted } = await res.json(); // { send: '0.01 ETH on Base', receive: '0.00028561
BTC', pair: 'ETH on Base → BTC' }
```

Python

```
import os, requests
r = requests.get("https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/price",
                 params={"from": "ETH on base", "to": "BTC", "amount": "0.01"},
                 headers={"x-maxy-key": os.environ["MAXY_API_KEY"]})
print(r.json()["formatted"]["receive"])  # 0.00028561 BTC
```

Response


```
{
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": {
    "chain": 8453,
    "chainName": "Base",
    "token": "0x0000000000000000000000000000000000000000",
    "symbol": "ETH",
    "decimals": 18,
    "caip2": "eip155:8453"
  },
  "to": {
    "chain": "bitcoin",
    "chainName": "Bitcoin",
    "token": "BTC",
    "symbol": "BTC",
    "decimals": 8,
    "caip2": "bip122:000000000019d6689c085ae165831e93"
  },
  "fromAmount": "10000000000000000",
  "toAmount": "28561",
  "toAmountMin": "28561",
  "formatted": {
    "send": "0.01 ETH on Base",
    "receive": "0.00028561 BTC",
    "pair": "ETH on Base → BTC"
  },
  "expiresAt": 1793644067,
  "limits": {
    "minSats": 546
  }
}
```


A real answer from the live API, 2 November 2026 (`x-request-id` 923847ec-52ad-4ecc-8a22-fb3d38f4a9a4).

Amounts are strings in the smallest unit (wei, sats, 6-decimal units); `formatted` has them in words. `toAmount` is what arrives, and it is the executable amount, so `toAmountMin` equals it. A price answers at any size.

Buying BTC, the answer also carries `limits.minSats`, the smallest order. The size of an order is checked on the quote for it: above your key's order size cap, `issues.orderCap` says "Order size is capped for this API key. Try $100 or less." and there is nothing to sign.

Every answer from the API says how long Maxy took in `Server-Timing: total;dur=…` (milliseconds; readable from a browser too) and carries an `x-request-id`. A key Maxy does not know is refused

before the API, with `x-request-id` only. `GET /` lists the main routes and the command that gets a test key; `/openapi.json` lists all of them.

10.1b Price or quote

Like other swap APIs, Maxy splits the two:

	`GET /price`	`GET /quote`
When	While the user browses or types an amount	When the user is about to order
Needs	The pair and an amount	Also `fromAddress` (who pays), `toAddress` (who receives) and your key
Gives	What arrives (`toAmount` , `formatted`)	The same amounts plus `quoteId` , `expiresAt` and the steps to sign and send
Reserves anything	No	No: the order does, when it is placed

Both answer at any size. Above your key's order size cap a quote still shows the amounts, with `issues.orderCap` ("Order size is capped for this API key. Try $100 or less.") and no steps, and `POST /orders` refuses the order with `order_cap_exceeded` . Nothing is signed.

10.1c Three ways to name a swap

Every price and quote takes whichever spelling your stack already uses:

Spelling	**Example**
Plain words, whole coins	`from=USDC on base&to=BTC&amount=100`
Route-aggregator style, smallest unit	`fromChain=8453&fromToken=0x8335…2913&toChain=bitcoin&toToken=BTC&fromAmount=100000000`
Swap-API style, smallest unit	`chainId=8453&sellToken=USDC&buyToken=BTC&sellAmount=100000000`

Coins in words: `ETH` (on Base), `USDC on arbitrum` , `arbitrum:USDC` , `BASE.ETH` , `BTC` , `SOL` . Chains: `base` , `arbitrum` , `bnb` , `avalanche` , `robinhood` , `bitcoin` , `solana` , a chain id, or a CAIP-2 id (`eip155:8453` , `bip122:000000000019d6689c085ae165831e93` , `solana:5eykt4UsFv8P8NJdTREpY1vzqKqZKvdp`). If your stack already numbers Bitcoin and Solana (Bitcoin `20000000000001` , `8253038` or `999999999994` ; Solana `1151111081099710` or `792703809`)

or names BTC by the placeholder address `bc1qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqmql8k8`, send them as they are. `GET /pairs` lists every pair.

10.1d Endpoints

Method and path	Key	What it does
`GET /`	no	This list as JSON, with the command that gets a test key
`POST /sandbox/keys`	no	A test key and a test publishable key, with no sign-up (the body can be left out, or `{ name, origins }`). Shown once
`GET /chains`	yes	Bitcoin, the EVM chains with a settler, and Solana
`GET /tokens?chain=base`	yes	Symbol, address and decimals of each token
`GET /pairs`	yes	Every pair, in words (`USDC on Base → BTC`), with its route, settler, smallest order, confirmations and CAIP-19 ids
`GET /price`	yes	What arrives for an amount; no wallet needed
`GET /prices/stream`	yes	The same question as server-sent events, every 5 seconds or more (or `?ticket=`, from a browser)
`POST /prices/stream-ticket`	yes	A ticket for that stream from a browser (`EventSource` cannot send a key): one pair, any amount, 10 minutes
`GET /quote`	yes	The quote for an order about to be placed (`fromAddress`, `toAddress`): `quoteId`, `expiresAt`, the steps and `signing`
`POST /quotes/batch`	yes	Up to 20 prices or quotes in one call, with an answer or an error for each
`POST /orders`	yes	The signed order: back comes the transaction to send (or, gasless, the solver takes it)
`POST /orders/submit`	yes	After the escrow or bridge transaction: tell Maxy (the solver also watches the chain)
`POST /orders/{id}/signature`	yes	The same as `POST /orders`, for the order in the path
`POST /orders/{id}/cancel`	yes	The transaction that takes back an escrow no solver committed to; `409 order_state` once one has
`GET /orders/{id}/events`	yes	The order's states so far, with times
`GET /orders/{id}/events/stream`	yes	The order's state as server-sent events, until it ends (or `?ticket=`, from a browser)
`POST /orders/{id}/stream-ticket`	yes	A ticket for that stream from a browser (`EventSource` cannot send a key): this order, 10 minutes, answers the URL to open

Method and path	Key	What it does
`GET /status`	yes	Where an order is, by `orderId` or by `txHash` (any transaction of the order)
`GET /orders/{id}`	yes	The same, by path
`GET /orders/{id}/refund`	yes	The transaction that takes an expired escrow back to the user, simulated (`chainId`)
`GET /orders/{id}/deposit`	yes	Selling BTC: where and how much to pay, read from the settler (`chainId`)
`GET /orders/{id}/claim`	yes	Selling BTC: the transaction that proves the payment and releases the payout, simulated (`chainId`)
`GET /orders/{id}/bridge`	yes	Selling BTC for a coin with no settler: the bridge transaction, after the settler has paid
`GET /orders/{id}/steps`	yes	Solana: where the offer is and what is left to sign
`POST /orders/{id}/steps`	yes	Solana: the next stage, or a signed transaction to send
`GET /orders`	yes	Your orders, newest first: `limit`, `cursor`, `from`, `to`, `state`, `wallet`, `externalId`, `externalCustomerId`; `format=csv` for a spreadsheet
`POST /orders/watch`	yes	Follow any order for your webhooks (`orderId`, `chainId?`, `route?`, `toChainId?`)
`POST /webhooks`	yes	Get signed `order.updated` events at your HTTPS endpoint (webhooks)
`GET /webhooks`	yes	Your endpoints, without their secrets
`DELETE /webhooks/{id}`	yes	Stop sending to an endpoint
`POST /webhooks/{id}/test`	yes	Send a signed `webhook.test` event now
`GET /webhooks/{id}/events`	yes	The last 25 events of an endpoint (`signed=1` adds each one signed now)
`POST /webhooks/{id}/events/{eventId}/replay`	yes	Send a past event again, with the same event id
`GET /key`	yes	Your key, its class, scopes, origins and order size cap
`GET /limits`	yes	Your key's class and order size cap, the smallest order per route, the quote lifetime and your rate limits right now
`GET /addresses/validate`	yes	Whether an address is well formed and a Maxy order can pay it (`chain`, `address`), with a reason
`GET /account`	yes	Your account: keys, usage and limit requests (a secret key)
`GET /usage`	yes	Your calls per day (`days=30`)

Method and path	Key	What it does
`POST /keys`	yes	A new publishable key `{ kind: "publishable", origins }` (with a test key, another test key). Shown once
`POST /keys/rotate`	yes	A new secret key with the same cap; the old one works until you revoke it
`POST /keys/{prefix}/origins`	yes	The sites a publishable key may be used from
`DELETE /keys/{prefix}`	yes	Revoke a key of your account
`POST /limits/requests`	yes	Ask for a higher order size cap `{ wantedUsd, note }`
`GET /limits/requests`	yes	What you asked for, and the answers
`GET /statements`	yes	A month of your orders with totals (`month=2026-11`, `format=json`, `csv` or `pdf`)
`GET /explorer/{query}`	yes	One order by order id, intent hash, chain transaction or Bitcoin txid: states, times and links
`GET /checkout/verify`	no	Hosted checkout: whether a signed URL is what the partner signed
`GET /system/status`	no	The status page: every component now, open incidents and maintenance (also `/system/history`, `/incidents`, `/maintenance`, `/verified`, `/feed.xml`)
`GET /health`	no	The solver, Bitcoin and each chain's verifier, with a `degraded` flag
`GET /openapi.json`	no	This API as OpenAPI 3.1
`GET /sandbox/scenarios`	test key	The outcomes you can force in the sandbox
`GET /sandbox/rates`	test key	Live BTC, ETH and USDC prices in dollars
`POST /sandbox/orders/{id}/advance`	test key	Move a sandbox order to its next state now, or to `{ "to": "completed" }`

Each route, with its parameters, samples in three languages and a real answer, is on the 10.2 API reference.

10.1e Buying BTC, end to end

EVM to BTC runs entirely over HTTP. Your user's wallet signs and sends; Maxy never holds a key.

1. `GET /quote` with the paying wallet (`fromAddress`) and the Bitcoin address (`toAddress`): `quoteId`, `order`, `issues` and `steps`. Each signature step carries the EIP-712 `typedData` and its `hash`.

2. The user's wallet signs each signature step (`eth_signTypedData_v4`, exactly as returned).
3. `POST /orders` with `{ order, signatures }`: the API rebuilds the order from its fields, re-reads the intent hash from the settler, re-derives the Bitcoin script from the address, checks the signatures came from the paying wallet, and simulates the escrow transaction from that wallet. Back come the `transaction` and its `simulation`.
4. The user's wallet sends the transaction; `POST /orders/submit` with `next.body` and its hash starts the solver at once (it also finds the escrow on chain by itself).
5. `GET /status` until `btcTxId` is set (or let a webhook tell you): from then the BTC is on its way.

Every step in curl, TypeScript and Python, with the real response at each, is on buy BTC from an EVM chain.

10.1f USDC to BTC with no gas

For USDC on Base and Arbitrum the quote returns signatures only: an EIP-2612 permit when the Permit2 allowance is short, and a Permit2 transfer bound to the order. `POST /orders` with `{ order, signatures: { permit?, witness } }` hands it to the solver, which opens the escrow and pays the gas. Your user never needs ETH, which is what a card on-ramp's users need. A token without an EIP-2612 permit gets an `approve` transaction for this order's amount instead of the permit; USDC on Base and Arbitrum never does. When the wallet holds less USDC than the order, the quote carries `issues.balance` and `quoteId`, `order` and `steps` are null.

10.1g Selling BTC, end to end

BTC to USDC, USDT or the native coin on Base, Arbitrum, BNB Chain, Avalanche or Robinhood runs entirely over HTTP. The solver locks the payout in the settler for your user before your user sends any BTC, and a Bitcoin proof releases it. Your user's wallet signs; Maxy never holds a key.

1. `GET /quote` with `fromAddress` (the Bitcoin address the user pays from) and `toAddress` (the EVM wallet that signs and gets paid; the settler pays no other): `quoteId`, `order` and `steps`. The signature step carries the EIP-712 `typedData` of the sell order and its `hash`, which is the order id. There is no Bitcoin address yet.
2. The user's wallet signs it (`eth_signTypedData_v4`, exactly as returned). No gas.
3. `POST /orders` with `{ order, signatures: { intent } }`. The API checks the order again, field by field, and that the signature came from `order.maker`; the solver then locks the payout. Back comes `orderId`, `escrow` (`locked`, `locking`, or `unconfirmed` with a `202` when the solver did not answer in time), `commitTx`, `lockBy`, `status` and `next`, the deposit route to poll.

4. `GET` `next.path` every few seconds. `{ "ready": false, "reason", "retryAfterSeconds" }` means the payout is not locked yet: **pay nothing**. `{ "ready": true, "address", "sats", "amountBtc", "bitcoinUri", "payout", "expiresAt", "btcDeadline", "blocksLeft", "requiredConfirmations", "instructions", "verify", "claim", "status" }` means it is locked. The address is decoded from the settler's own stored script, never from a server's text, and `verify` says how to read the escrow yourself.
5. The user pays exactly `sats` to `address`, in one output, once. A different amount, a split payment or a payment after `btcDeadline` cannot be paid out.
6. `GET /status` (or a webhook) until `completed`. The solver sends the proof itself. If it does not, `GET /orders/{orderId}/claim?chainId=…` answers the proof as a simulated transaction any wallet can send, or `{ "claimable": false, "reason", "retryable" }` before the payment has its confirmations.

A wallet may have three unpaid sell orders open.

A coin no settler pays (ETH or USDC on Ethereum, for example) is two legs, neither through Maxy's custody: the settler pays the native coin on Base to the user's wallet, then that wallet bridges it. The quote's `toAmount` and `toAmountMin` are the coin asked for, as the bridge prices it now, and its steps end with `bridge`. After the settler has paid, `GET` that path: `{ "ready": true, "provider", "expiresAt", "transaction", "simulation", "toAmount", "toAmountMin" }` for the wallet to send before `expiresAt` (after it, `GET` again; it keeps a little of the payout for gas), or `{ "ready": false, "reason" }` before. The route is the solver's route picker's, checked as in EVM to EVM below.

10.1h SOL or an SPL token to BTC, end to end

Also entirely over HTTP. The coins sit in the Solana settler's escrow until the Bitcoin payment is proven on Solana; Maxy's servers never hold them, and the user's wallet signs everything.

1. `GET /quote` with `fromAddress` (the Solana wallet) and `toAddress` (a `bc1q` or `bc1p` address): `quoteId`, `order`, `issues` and `steps`. The first step is `kind: "signature"`, `signatureKind: "solana-message"`: the UTF-8 `message` the wallet signs with `signMessage`. It names the wallet, the amount and the Bitcoin address, and lasts 4 minutes. Nothing moves.
2. `POST /orders` with `{ order, signatures: { quote } }` (base64; base58 and hex also read). The market maker opens its offer on Solana, about 20 seconds. Back come `orderId`, `order` (the offer's fields added: send this one from now on), `expiresAt` and `steps`: the deposit `transaction`, unsigned, in base64. A `202` with `transaction: null` means the offer is not readable on Solana yet: `POST` `next` in a few seconds.
3. The wallet signs the transaction (`signTransaction`; it must not change it).

4. `POST /orders/{orderId}/steps` with `{ order, signatures, transaction }`. The transaction must be signed by the order's wallet and be, byte for byte apart from that signature, the deposit Maxy builds for the order from the offer on Solana now. Only then is it sent, after a simulation. Back come `submitted`, `signature` and `status`.
5. `GET /status?orderId=…&route=sol-btc` (or a webhook) until `completed`. The market maker pays the BTC and proves it; if it never pays, the escrow goes back to the wallet.

The deposit's blockhash lasts about a minute. `POST /orders/{orderId}/steps` with `{ order, signatures }` and no `transaction` gives a fresh one to sign. `GET /orders/{orderId}/steps` says where the offer is and what is left. An offer more than 0.1% below the quote builds no deposit (`409 quote_mismatch`): get a new quote.

10.1i Refunds

`GET /orders/{orderId}/refund?chainId=…` answers `{ "refundable": true, "function", "explanation", "transaction", "simulation" }` for the user's wallet to send, or `{ "refundable": false, "reason", "unlocksAt", "transaction": null }`. 5.10 Refunds and failures has every case and what to tell the user.

10.1j BTC to SOL or an SPL token, end to end

The app's own flow, split at each wallet prompt. The user's wallet opens an offer on the Solana settler; the market maker fills it by escrowing the coin in the settler; the user's Bitcoin payment is staged and locked on the settler; after a confirmation the market maker proves the payment on Solana and settles, and the settler pays the user's wallet (it pays only the offer's maker, and only against the proof). The market maker builds and pays for every Solana transaction: the Solana wallet signs (never sends) and needs no SOL. It needs a SegWit (`bc1q`) or Taproot (`bc1p`) Bitcoin wallet. **The BTC goes out last**: the signed payment is not broadcast by the wallet; the market maker sends it once it is locked.

1. `GET /quote` with `fromAddress` (the address that pays) and `toAddress` (the Solana wallet): `order`, `issues`.
2. `POST /orders` with `{ order }`: the market maker builds the open at its price now, and one more than 0.1% below the quote is refused before anything is signed (`409 quote_mismatch`). Back come `orderId`, `order` and the open to sign.

From here every call is `POST /orders/{orderId}/steps` with `{ order, stage, … }`, always sending the latest `order`. Each answer's `steps` and `next` say exactly what to do. Every Solana transaction is checked before it is offered and again when it comes back signed: the market maker

pays the fee, the wallet is a required signer, only the settler's programs and Solana's own run, and nothing spends from the wallet.

`stage`	What it does	What the wallet does
`fill` (with `transactions`)	The market maker sends the signed open and fills the offer. Back comes the Bitcoin payment as a PSBT, built exactly as the fill's plan says (checked against the fill on Solana). `202` while a larger order fills: send `fill` again without `transactions`	`signTransaction` on the open, then signs the PSBT (`inputsToSign`) and **does not broadcast it**
`pay`	The PSBT again, for a filled offer	Signs it
`lock` (with `signedPsbt`)	Checks the payment pays only what the plan says; back comes the stage + lock transaction	`signTransaction`
`submit` (with `transactions`)	The market maker sends the stage + lock, then the BTC. It proves and settles after a confirmation	Nothing more
`cancel`	An offer the market maker did not fill: the close to sign, then `cancel` again with `transactions`	`signTransaction`
`state`	Reads the offer from Solana and from the market maker (`phase`, `settleSignature`) and names the next stage	Nothing

If the market maker does not fill the offer, `fill` answers `409` with the `cancel` step; nothing has moved. If the user never pays, its escrow expires back to it. A `submit` that reaches Solana too late answers `quote_expired`: send `lock` again.

10.1k EVM to EVM, end to end

Bridged on the solver's route picker's route (of Mayan, deBridge, Across and LI.FI, the one that pays out most after every fee), because no settler serves these pairs. The coins stay in the user's wallet until it sends them, and the receiver is checked in the calldata. The order names its `provider` and the transaction's `expiresAt`.

1. `GET /quote` with `fromAddress` and `toAddress`: `quoteId`, `order`, `issues` (`balance`, `allowance`) and `steps`. When the allowance is short, an `approve` transaction for exactly this amount comes first.
2. `POST /orders` with `{ order }`: the transaction is checked again by its provider's rules (the provider's contract, the receiver, the destination chain, the coin and amount, the minimum, its expiry) and simulated from the wallet. Back come `transaction` and `simulation`. Past the transaction's `expiresAt` the answer is `409 quote_expired` with a fresh quote.

3. The wallet sends it. `POST /orders/submit` with `{ order, txHash }` files it for status and webhooks.

Your key's order size cap applies here too, at the dollars the bridge values the amount at: the quote still shows the amounts, with `issues.orderCap` and no steps, and `POST /orders` answers `422 order_cap_exceeded`.

10.1l Quotes last 60 seconds

`expiresAt` is 60 seconds after the quote, and `POST /orders` enforces it: an order from an older quote is `409 quote_expired`, and nothing was sent. The order carries a stamp (`order.quote`) only Maxy can make, so send the order back unchanged. `POST /orders/submit` is never refused for age, because the funds have already moved.

10.1m Buying an exact amount of BTC

Give what you want instead of what you send: `receive=1.31` (whole BTC) or `toAmount=131000000` (sats), in place of `amount` or `fromAmount`. It works for buying BTC from any EVM coin or from Solana, on `/price` and `/quote`. The answer's `fromAmount` is the smallest amount that buys that many sats at the same price a quote by amount gets. Other pairs answer `400`: their quotes are not sized this way.

10.1n Status

`GET /status?orderId=…&chainId=…&route=…` (route defaults to `evm-btc-escrow`), or `GET /orders/{orderId}`, returns `{ orderId, route, state, rawStatus, isComplete, btcTxId, txs }`. Holding only a transaction hash (the escrow or settlement transaction, or the Bitcoin payment's txid)? `GET /status?txHash=…` finds the order. `state` is one of `pending`, `awaiting_deposit`, `in_progress`, `completed`, `refundable`, `refunded`, `cancelled`, `expired`, `failed` (order lifecycle).

Orders placed with your key, through this API or the SDK, are filed under it: `GET /orders` lists them with their state (add `format=csv` for your finance team), and their changes reach your webhooks.

10.1o Webhooks

Instead of polling `/status`, register an HTTPS endpoint and Maxy calls it each time one of your orders changes state. Orders placed through this API, and orders the SDK places with your key, are followed automatically; any other order is followed after `POST /orders/watch`.

Register an endpoint. The answer carries the endpoint's signing secret, shown this once: store it.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" -H "content-type: application/json"
"$BASE/webhooks" \
  -d '{ "url": "https://your.app/maxy/webhooks" }'
```

Response

```
{ "id": "5b7c…", "url": "https://your.app/maxy/webhooks", "events": ["order.updated"], "secret":
"whsec_…", "createdAt": "2026-11-01T18:20:00Z" }
```

What arrives. A `POST` with a JSON body, sent when the order's `state` or its `btcTxId` changes:

Response

```
{
  "id": "evt_3f9c2a…",
  "type": "order.updated",
  "createdAt": "2026-11-01T18:21:07.412Z",
  "data": {
    "orderId": "0x5c1e…", "route": "evm-btc-escrow", "chainId": 8453,
    "state": "completed", "previousState": "in_progress", "rawStatus": "BTC_SENT",
    "isComplete": true, "btcTxId": "f3a1…9c0e", "txs": [{ "label": "btc_payout", "hash": "f3a1…
9c0e" }]
  }
}
```

Headers: `Maxy-Signature: t=<unix seconds>,v1=<hex>`, `Maxy-Event-Id`, `Maxy-Event-Type`. `v1` is HMAC-SHA256 of `<t>.<raw body>` keyed with the endpoint secret.

Check every delivery against the raw body, before parsing it. With the SDK:

TypeScript

```
import { verifyWebhook } from '@maxy-labs/sdk';

app.post('/maxy/webhooks', express.raw({ type: 'application/json' }), async (req, res) => {
  const event = await verifyWebhook(req.body.toString(), req.header('maxy-signature'),
process.env.MAXY_WEBHOOK_SECRET);
  // event.data.state, event.data.btcTxId …
  res.sendStatus(200);
});
```

Without it, in any language (Python here):

Python

```
import hmac, hashlib, time

def verify(raw_body: bytes, header: str, secret: str, tolerance=300) -> bool:
    parts = dict(p.split("=", 1) for p in header.split(","))
    t, sig = int(parts["t"]), parts["v1"]
    expected = hmac.new(secret.encode(), f"{t}.".encode() + raw_body,
hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, sig) and abs(time.time() - t) <= tolerance
```

Delivery. Answer with any 2xx within 10 seconds. Anything else is retried after 30 s, 2 min, 10 min, 30 min, 1 h, 3 h, 6 h and 12 h, then marked failed (9 attempts over about 23 hours). Delivery is at least once, so dedupe on the event `id`. Up to 8 events go out at once, so events can arrive out of order: order them by `createdAt`, or compare `previousState`. Changes are picked up within about 15 seconds; an order is followed until it is final, or for 3 days.

Method and path	What it does
`POST /webhooks`	Register an endpoint (`{ url }`, https on a public host; up to 5 per key)
`GET /webhooks`	Your endpoints (never the secrets)
`DELETE /webhooks/{id}`	Stop sending to an endpoint
`POST /webhooks/{id}/test`	Send a signed `webhook.test` event now; answers with your endpoint's HTTP status
`GET /webhooks/{id}/events`	The last 25 events for an endpoint, with attempts and the last answer
`POST /orders/watch`	Follow any order: `{ orderId, chainId?, route?, toChainId? }`

10.1p Your account and the status page

Route	Does
`POST /sandbox/keys`	A test key and a test publishable key, with no sign-up. Shown once
`GET /account`	Your keys by prefix, usage, limit requests
`GET /usage`	Calls per day
`POST /keys`, `POST /keys/rotate`	A publishable key, or a new secret key with the same cap

Route	Does
`POST /keys/{prefix}/origins`, `DELETE /keys/{prefix}`	A publishable key's origins; revoking
`POST /limits/requests`, `GET /limits/requests`	Asking for a higher cap, and the answers
`GET /statements`	A month's orders with totals: JSON, CSV or PDF
`GET /checkout/verify`	4.7 Hosted checkout: is this signed link what you signed
`GET /explorer/{query}`	An order by id, intent hash, transaction or Bitcoin txid
`POST /feedback`, `POST /security-reports`	Docs feedback; a security report
`GET /system/status`, `GET /system/history`, `GET /system/incidents`, `GET /system/maintenance`, `GET /system/verified`, `GET /system/feed.xml`	The data behind maxy.co/status
`POST /system/subscribe`, `GET /system/unsubscribe`, `GET /system/confirm`	Status notices

10.1q Errors

Every error is JSON with a stable `code` to branch on and a `message` to show, plus the request's id (also in the `x-request-id` header):

Response

```
{ "error": { "code": "invalid_input", "message": "Say which chain PEPE is on, for example \"PEPE on base\" (base, arbitrum, bnb, avalanche, robinhood, bitcoin or solana).", "docs": "https://maxy.co/docs/sdk-errors" }, "requestId": "6f0c…" }
```

HTTP	Codes
400	`invalid_input`, `unsupported_route`, `unsupported_chain`, `wallet_mismatch`, `verification_failed`
401	`auth_required`, `invalid_api_key`
403	`key_revoked`, `sandbox_key_refused`
404	`not_found`
409	`quote_expired`, `quote_mismatch`, `conflict`
422	`amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`, `deposit_address_unverified`, `server_rejected`

HTTP	Codes
429	`rate_limited` (with a `Retry-After` header)
500, 501	`internal_error`, `not_implemented`
502	`bad_response`, `http_error`, `network_error`
503, 504	`route_disabled`, `timeout`

Each code's cause, whether to retry, and what to do: HTTP API error codes.

10.1r Limits

Rate limits. 600 calls a minute per key, wherever they come from (your backend can serve every user from one address). Every answer carries `RateLimit-Limit`, `RateLimit-Remaining` and `RateLimit-Reset` (seconds); over the limit is `429 rate_limited` with `Retry-After`. Ask Maxy for more.

Test keys. 120 calls a minute per key, and 50 sandbox orders open at a time.

Order size cap. Your key's cap applies to orders, never to prices or quotes (limits). Every call made with a key is metered under it.

10.1s Versions and changes

Routes carry no version. The API only gains fields and routes; nothing is removed or renamed. A breaking change is announced to every key holder at least 90 days ahead, with a migration guide, and the old behaviour keeps working for at least six months. The changelog lists every change.

10.1t Where next

- Integration guide: what Maxy is, the quickstart, and the flows step by step.
- Swap guide: what each route does underneath.
- Errors, limits, security.

Next: 10.2 API reference.

Last verified: 7 November 2026, against the SDK source and the API it runs.

10.2 API reference

After this page you can call any route of the HTTP API: its parameters, samples in curl, TypeScript and Python, a real answer, and its errors.

Before you start: a test key, and `BASE` and `KEY` from the setup. **Time:** look up what you need.

This page is generated from `GET /openapi.json`, and every answer is a real sandbox capture. The API only gains fields, so ignore what you do not know. The 10.1 HTTP API page tells the routes as flows, and has a live panel to try a price.

10.2a GET /

Every route, and the command that gets a test key. No key needed.

curl

```
curl -s "$BASE"
```

TypeScript

```
const res = await fetch(`${BASE}`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}").json()
```

Response


```
{
  "name": "Maxy API",
  "sdk": "0.18.0",
  "baseUrl": "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api",
  "try": "curl -s -X POST https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-
api/sandbox/keys",
  "docs": "https://maxy.co/docs/first-order",
  "openapi": "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/openapi.json",
  "auth": "Every route takes your key in x-maxy-key (or Authorization: Bearer mxk_live_…), apart
from /health, /openapi.json, the status page (/system/*) and POST /sandbox/keys, which gives you
a test key.",
  "endpoints": [
    {
      "route": "GET /chains",
      "key": true,
      "what": "Chains Maxy swaps on"
    },
    {
      "route": "GET /tokens",
      "key": true,
      "what": "Tokens per chain (?chain=base)"
    },
    "…"
  ]
}
```


`GET` : a real sandbox answer, 13 November 2026 (`x-request-id` 8277cf47-b037-45e7-9d73-56c7bcc5b9d0), with long fields folded.

Fields: `name`, `sdk`, `baseUrl`, `try`, `docs`, `openapi`, `auth`, `endpoints`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2b GET /openapi.json

This API as OpenAPI 3.1. No key needed.

curl

```
curl -s "$BASE/openapi.json" | jq .info.title
```

TypeScript

```
const spec = await fetch(`${BASE}/openapi.json`).then((r) => r.json());
```

Python

```
spec = requests.get(f"{BASE}/openapi.json").json()
```

Response


```
{
  "openapi": "3.1.0",
  "info": {
    "title": "Maxy API",
    "version": "stable",
    "description": "…"
  },
  "servers": [
    {
      "url": "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api"
    }
  ],
  "webhooks": {
    "order.updated": {
      "post": {
        "summary": "An order changed state",
        "description": "Signed: Maxy-Signature: t=<unix>,v1=<hex HMAC-SHA256(secret, \"<t>.<raw body>\")>. Answer 2xx within 10 s; retried with backoff for about 22 hours. Dedupe on id.",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "properties": {
                  "id": {
                    "type": "string"
                  },
                  "type": {
                    "type": "string"
                  },
                  "createdAt": {
                    "type": "string"
                  },
                  "data": {
                    "type": "object"
                  }
                }
              }
            }
          }
        },
        "responses": {
          "200": {
            "description": "Received."
          }
        }
      }
```

```
    }
  },
  "components": "…",
  "security": [
    {
      "apiKey": []
    }
  ],
  "paths": "…"
}
```

`GET /openapi.json`: a real sandbox answer, 13 November 2026 (`x-request-id` f2c724d9-b777-46b5-8fa4-963b623ae3d9), with long fields folded.

Fields: `openapi`, `info`, `servers`, `webhooks`, `components`, `security`, `paths`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2c GET /chains

Chains Maxy swaps on. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/chains"
```

TypeScript

```
const res = await fetch(`${BASE}/chains`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/chains", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "chains": [
    {
      "chain": "bitcoin",
      "name": "Bitcoin",
      "type": "bitcoin",
      "nativeSymbol": "BTC",
      "settler": null,
      "explorer": "https://mempool.space",
      "caip2": "bip122:000000000019d6689c085ae165831e93"
    },
    {
      "chain": 8453,
      "name": "Base",
      "type": "evm",
      "nativeSymbol": "ETH",
      "settler": "0xeC5a05b28dd848F628e19CB26Df32f053e64ab86",
      "explorer": "https://basescan.org",
      "caip2": "eip155:8453"
    },
    "…"
  ],
  "sandbox": true
}
```


`GET /chains`: a real sandbox answer, 13 November 2026 (`x-request-id` a91d0dd7-3986-4030-b99b-522a86ea8d4a), with long fields folded.

Fields: `chains`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2d GET /tokens

Tokens per chain. Needs your key.

Parameter	Required	Meaning
`chain`	no	A chain name, id or CAIP-2 id: base, 8453, eip155:8453, bitcoin, bip122:000000000019d6689c085ae165831e93, solana, solana:5eykt4UsFv8P8NJdTREpY1vzqKqZKvdp.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/tokens?chain=base"
```

TypeScript

```
const res = await fetch(`${BASE}/tokens?${new URLSearchParams({ chain: 'base' })}`, { headers: {
'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/tokens", headers={"x-maxy-key": KEY}, params={"chain":
"base"}).json()
```

Response


```
{
  "tokens": [
    {
      "chain": 8453,
      "chainName": "Base",
      "symbol": "ETH",
      "address": "0x0000000000000000000000000000000000000000",
      "decimals": 18,
      "assetId": "eip155:8453/slip44:60"
    },
    {
      "chain": 8453,
      "chainName": "Base",
      "symbol": "USDC",
      "address": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
      "decimals": 6,
      "assetId": "eip155:8453/erc20:0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913"
    },
    "…"
  ],
  "sandbox": true
}
```


`GET /tokens`: a real sandbox answer, 13 November 2026 (`x-request-id` b31d8fb7-6c7f-452c-a327-52658715851f), with long fields folded.

Fields: `tokens`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2e GET /pairs

Every pair, with its route, settler, smallest order and confirmations. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/pairs"
```

TypeScript

```
const res = await fetch(`${BASE}/pairs`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/pairs", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "pairs": [
    {
      "from": {
        "chain": 8453,
        "token": "0x0000000000000000000000000000000000000000",
        "symbol": "ETH",
        "decimals": 18,
        "chainName": "Base"
      },
      "to": {
        "chain": "bitcoin",
        "token": "BTC",
        "symbol": "BTC",
        "decimals": 8,
        "chainName": "Bitcoin"
      },
      "label": "ETH on Base → BTC",
      "fromAssetId": "eip155:8453/slip44:60",
      "toAssetId": "bip122:000000000019d6689c085ae165831e93/slip44:0",
      "route": "evm-btc-escrow",
      "settler": "0xeC5a05b28dd848F628e19CB26Df32f053e64ab86",
      "minSats": 546,
      "requiredConfirmations": 1,
      "confirmationsBySize": false
    },
    {
      "from": {
        "chain": 8453,
        "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
        "symbol": "USDC",
        "decimals": 6,
        "chainName": "Base"
      },
      "to": {
        "chain": "bitcoin",
        "token": "BTC",
        "symbol": "BTC",
        "decimals": 8,
        "chainName": "Bitcoin"
      },
      "label": "USDC on Base → BTC",
      "fromAssetId": "eip155:8453/erc20:0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
      "toAssetId": "bip122:000000000019d6689c085ae165831e93/slip44:0",
      "route": "evm-btc-gasless",
      "settler": "0xeC5a05b28dd848F628e19CB26Df32f053e64ab86",
      "minSats": 546,
```

```
      "requiredConfirmations": 1,
      "confirmationsBySize": false
    },
    "…"
  ],
  "sandbox": true
}
```

`GET /pairs`: a real sandbox answer, 13 November 2026 (`x-request-id` 0a46da56-61c4-4911-8f16-de025ca1482c), with long fields folded.

Fields: `pairs`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2f GET /addresses/validate

Is an address valid for a chain, and would an order pay it. Needs your key.

Parameter	Required	Meaning
`chain`	yes	A chain name, id or CAIP-2 id: base, 8453, eip155:8453, bitcoin, bip122:000000000019d6689c085ae165831e93, solana, solana:5eykt4UsFv8P8NJdTREpY1vzqKqZKvdp.
`address`	yes	

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/addresses/validate?
chain=bitcoin&address=bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq"
```

TypeScript

```
const res = await fetch(`${BASE}/addresses/validate?${new URLSearchParams({ chain: 'bitcoin',
address: 'bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/addresses/validate", headers={"x-maxy-key": KEY}, params=
{"chain": "bitcoin", "address": "bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq"}).json()
```

Response

```
{
  "address": "bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq",
  "chain": "bitcoin",
  "valid": true,
  "type": "p2wpkh",
  "network": "mainnet",
  "payable": true,
  "reason": null,
  "sandbox": true
}
```

`GET /addresses/validate`: a real sandbox answer, 13 November 2026 (`x-request-id` 21ef731c-42d2-446c-9ff7-28a25f97f02a).

Fields: `address`, `chain`, `valid`, `type`, `network`, `payable`, `reason`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2g GET /limits

Your key's class and order size cap, the smallest order per route, how long a quote lasts, and your rate limits. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/limits"
```

TypeScript

```
const res = await fetch(`${BASE}/limits`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/limits", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "kind": "secret",
  "scopes": [
    "read",
    "quote",
    "place",
    "orders",
    "webhooks",
    "account"
  ],
  "origins": [],
  "orderCapUsd": 100,
  "minOrderSats": 546,
  "routes": {
    "evm-btc-escrow": {
      "minSats": 546
    },
    "evm-btc-gasless": {
      "minSats": 546
    },
    "btc-evm": {
      "minUsd": 10
    },
    "sol-btc": {
      "minUsd": 2
    },
    "btc-sol": {
      "minUsd": 2
    }
  },
  "quoteTtlSeconds": 60,
  "batchMax": 20,
  "rateLimit": {
    "limit": 120,
    "remaining": 118,
    "resetSeconds": 60,
    "per": "key",
    "priceLimit": 1200
  },
  "sandbox": true
}
```


`GET /limits`: a real sandbox answer, 13 November 2026 (`x-request-id` 2a9d335f-ceff-43fc-9c3d-5b741f61a402).

Fields: `kind`, `scopes`, `origins`, `orderCapUsd`, `minOrderSats`, `routes`, `quoteTtlSeconds`, `batchMax`, `rateLimit`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2h GET /health

The solver, Bitcoin and each chain's verifier, with a degraded flag. No key needed.

curl

```
curl -s "$BASE/health"
```

TypeScript

```
const res = await fetch(`${BASE}/health`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/health").json()
```

Response


```
{
  "status": "ok",
  "degraded": false,
  "checkedAt": "2026-11-13T02:04:27.415Z",
  "components": {
    "api": {
      "status": "ok"
    },
    "quotes": {
      "status": "ok",
      "latencyMs": 158
    },
    "solver": {
      "status": "ok",
      "latencyMs": 158,
      "priceAgeSeconds": 8
    },
    "bitcoin": {
      "status": "ok",
      "tipHeight": 975617
    },
    "verifiers": [
      {
        "chainId": 8453,
        "chain": "Base",
        "status": "ok",
        "height": 975617,
        "lagBlocks": 0
      },
      {
        "chainId": 56,
        "chain": "BNB Chain",
        "status": "ok",
        "height": 975617,
        "lagBlocks": 0
      },
      {
        "chainId": 42161,
        "chain": "Arbitrum",
        "status": "ok",
        "height": 975617,
        "lagBlocks": 0
      },
      {
        "chainId": 43114,
        "chain": "Avalanche",
        "status": "ok",
```

```
        "height": 975617,
        "lagBlocks": 0
      },
      {
        "chainId": 4663,
        "chain": "Robinhood",
        "status": "ok",
        "height": 975617,
        "lagBlocks": 0
      }
    ]
  }
}
```


`GET /health`: a real sandbox answer, 13 November 2026 (`x-request-id` f37adee8-d4df-48d6-a6e2-2023eeecac80).

Fields: `status`, `degraded`, `checkedAt`, `components`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2i GET /key

Your key, its class and its order size cap. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/key"
```

TypeScript

```
const res = await fetch(`${BASE}/key`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/key", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "key": "mxk_test_8GOVco…",
  "kind": "secret",
  "scopes": [
    "read",
    "quote",
    "place",
    "orders",
    "webhooks",
    "account"
  ],
  "origins": [],
  "orderCapUsd": 100,
  "sandbox": true
}
```


`GET /key`: a real sandbox answer, 13 November 2026 (`x-request-id` 104dbd34-c9fd-41e7-99ff-daa1f12c9c3b).

Fields: `key`, `kind`, `scopes`, `origins`, `orderCapUsd`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2j GET /price

Check prices: what an amount gets right now. No wallet. Needs your key.

Parameter	Required	Meaning
`from`	no	What you sell, in words: "ETH", "USDC on arbitrum", "BASE.ETH", "BTC", "SOL", "USDC on eip155:8453". Or use fromChain + fromToken.
`to`	no	What you buy, in words. A CAIP-2 id alone is its chain's own coin (bip122:000000000019d6689c085ae165831e93 is BTC). Or use toChain + toToken.
`amount`	no	How much you sell, in whole coins. Or use fromAmount.
`fromChain`	no	Chain id, name or CAIP-2 id (8453, base, eip155:8453, bitcoin, solana). Alias: chainId.
`fromToken`	no	Token address or symbol. Alias: sellToken.
`toChain`	no	Chain id, name or CAIP-2 id; defaults to bitcoin when toToken is BTC.
`toToken`	no	Token address or symbol. Alias: buyToken.
`fromAmount`	no	How much you sell, in the smallest unit. Alias: sellAmount.

Parameter	Required	Meaning
`scenario`	no	Sandbox only: forces an outcome with a test key (GET /sandbox/scenarios lists them). A live key ignores it.
`simulate`	no	Sandbox only: the same as scenario, also by other names: quote_expired, fill_timeout, underpaid, refund.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/price?from=USDC%20on%20base&to=BTC&amount=10"
```

TypeScript

```
const res = await fetch(`${BASE}/price?${new URLSearchParams({ from: 'USDC on base', to: 'BTC',
amount: '10' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/price", headers={"x-maxy-key": KEY}, params={"from": "USDC on
base", "to": "BTC", "amount": "10"}).json()
```

Response


```
{
  "route": "evm-btc-gasless",
  "liquidityAvailable": true,
  "from": {
    "chain": 8453,
    "chainName": "Base",
    "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
    "symbol": "USDC",
    "decimals": 6,
    "caip2": "eip155:8453",
    "assetId": "eip155:8453/erc20:0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913"
  },
  "to": {
    "chain": "bitcoin",
    "chainName": "Bitcoin",
    "token": "BTC",
    "symbol": "BTC",
    "decimals": 8,
    "caip2": "bip122:000000000019d6689c085ae165831e93",
    "assetId": "bip122:000000000019d6689c085ae165831e93/slip44:0"
  },
  "fromAmount": "10000000",
  "toAmount": "13667",
  "toAmountMin": "13667",
  "formatted": {
    "send": "10.00 USDC on Base",
    "receive": "0.00013667 BTC",
    "pair": "USDC on Base → BTC"
  },
  "expiresAt": 1794535530,
  "expiresAtIso": "2026-11-13T02:05:30.000Z",
  "limits": {
    "minSats": 546
  },
  "sandbox": true
}
```


`GET /price`: a real sandbox answer, 13 November 2026 (`x-request-id` a4e23a42-1909-467e-acc7-5a24f4d66c99).

Fields: `route`, `liquidityAvailable`, `from`, `to`, `fromAmount`, `toAmount`, `toAmountMin`, `formatted`, `limits`, `expiresAt`, `expiresAtIso`, `message`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2k GET /quote

The quote for an order about to be placed: id, expiry and the steps to execute it. Needs your key.

Parameter	Required	Meaning
`from`	no	What you sell, in words: "ETH", "USDC on arbitrum", "BASE.ETH", "BTC", "SOL", "USDC on eip155:8453". Or use fromChain + fromToken.
`to`	no	What you buy, in words. A CAIP-2 id alone is its chain's own coin (bip122:000000000019d6689c085ae165831e93 is BTC). Or use toChain + toToken.
`amount`	no	How much you sell, in whole coins. Or use fromAmount.
`fromChain`	no	Chain id, name or CAIP-2 id (8453, base, eip155:8453, bitcoin, solana). Alias: chainId.
`fromToken`	no	Token address or symbol. Alias: sellToken.
`toChain`	no	Chain id, name or CAIP-2 id; defaults to bitcoin when toToken is BTC.
`toToken`	no	Token address or symbol. Alias: buyToken.
`fromAmount`	no	How much you sell, in the smallest unit. Alias: sellAmount.
`scenario`	no	Sandbox only: forces an outcome with a test key (GET /sandbox/scenarios lists them). A live key ignores it.
`simulate`	no	Sandbox only: the same as scenario, also by other names: quote_expired, fill_timeout, underpaid, refund.
`fromAddress`	yes	The paying wallet. Alias: taker.
`toAddress`	yes	Where the coins go (a bc1q… address when buying BTC). Alias: recipient.
`slippageBps`	no	
`receive`	no	Buying BTC: the BTC wanted, in whole BTC, instead of amount.
`toAmount`	no	Buying BTC: the sats wanted, instead of amount. Alias: buyAmount.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/quote?
from=ETH%20on%20base&to=BTC&amount=0.004&fromAddress=$WALLET&toAddress=$MY_BTC_ADDRESS"
```

TypeScript

```
const res = await fetch(`${BASE}/quote?${new URLSearchParams({ from: 'ETH on base', to: 'BTC',
amount: '0.004', fromAddress: account.address, toAddress: btcAddress })}`, { headers: { 'x-maxy-
key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, params={"from": "ETH on base", "to": "BTC", "amount": "0.004", "fromAddress": account.address, "toAddress": btc_address}).json()
```

Response


```
{
  "route": "evm-btc-escrow",
  "liquidityAvailable": true,
  "from": "…",
  "to": "…",
  "fromAmount": "4000000000000000",
  "toAmount": "10887",
  "toAmountMin": "10887",
  "formatted": {
    "send": "0.004 ETH on Base",
    "receive": "0.00010887 BTC",
    "pair": "ETH on Base → BTC"
  },
  "expiresAt": 1794535550,
  "expiresAtIso": "2026-11-13T02:05:50.000Z",
  "limits": {
    "minSats": 546
  },
  "quoteId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "order": {
    "route": "evm-btc-escrow",
    "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
    "chainId": 8453,
    "btcAddress": "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4",
    "intent": "…",
    "status": "/status?
orderId=0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3&chainId=8453&route=evm-
btc-escrow",
    "sandbox": "…",
    "quote": {
      "expiresAt": 1794535550,
      "id": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
      "token": "e2860bd87c250703d6079b481f847a2ae01f097e7fca558b97b51a7aced769a3",
      "ask":
"from=ETH+on+base&to=BTC&amount=0.004&fromAddress=0xdd0d268Abc55f328F88f1Ee2e2f0850bFf43686b&toAddr
    }
  },
  "issues": {
    "balance": null,
    "allowance": null,
    "orderCap": null
  },
  "steps": [
    {
      "id": "intent",
      "kind": "signature",
      "description": "Sign the order (EIP-712). No gas. Sandbox: this signature authorizes
```

```
nothing.",
      "signatureKind": "eip712",
      "typedData": {
        "domain": {
          "name": "Maxy Sandbox",
          "version": "1",
          "chainId": 8453,
          "verifyingContract": "0x0000000000000000000000000000000000000000"
        },
        "types": "…",
        "primaryType": "MaxySandboxOrder",
        "message": {
          "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
          "step": "intent",
          "notice": "Maxy sandbox. This moves nothing and cannot be spent."
        }
      },
      "hash": "0x25b9b9f55bfdf01ce7a847c7342f934e59e5c02d58ab9d8af87a17f1cba4773c"
    },
    {
      "id": "order",
      "kind": "post",
      "description": "Send the order and its signature; you get back the transaction to send.",
      "method": "POST",
      "path": "/orders",
      "body": "…"
    },
    {
      "id": "escrow",
      "kind": "transaction",
      "description": "Sandbox: do not send the transaction. POST /orders/submit with its
simulatedHash.",
      "chainId": 8453,
      "transaction": null
    }
  ],
  "signing": {
    "kind": "eip712",
    "payloads": [
      {
        "id": "intent",
        "typedData": {
          "domain": {
            "name": "Maxy Sandbox",
            "version": "1",
            "chainId": 8453,
            "verifyingContract": "0x0000000000000000000000000000000000000000"
          },
          "types": {
```

```
            "MaxySandboxOrder": [
              {
                "name": "orderId",
                "type": "string"
              },
              {
                "name": "step",
                "type": "string"
              },
              {
                "name": "notice",
                "type": "string"
              }
            ]
          },
          "primaryType": "MaxySandboxOrder",
          "message": {
            "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
            "step": "intent",
            "notice": "Maxy sandbox. This moves nothing and cannot be spent."
          }
        },
        "hash": "0x25b9b9f55bfdf01ce7a847c7342f934e59e5c02d58ab9d8af87a17f1cba4773c"
      }
    ],
    "approval": null,
    "after": {
      "method": "POST",
      "path": "/orders"
    }
  },
  "sandbox": true
}
```


`GET /quote`: a real sandbox answer, 13 November 2026 (`x-request-id` 6503b124-ab52-4f9b-a6dd-3bcac3d29a36), with long fields folded.

Fields: `route`, `liquidityAvailable`, `from`, `to`, `fromAmount`, `toAmount`, `toAmountMin`, `formatted`, `limits`, `expiresAt`, `expiresAtIso`, `message`, `quoteId`, `order`, `issues`, `steps`, `signing`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2l POST /orders

Send the signed order. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders" -H "content-type:
application/json" \
  -d "$(jq -c --arg sig "$SIG" '{order: .order, signatures: {intent: $sig}}' quote.json)"
```

TypeScript

```
const res = await fetch(`${BASE}/orders`, { method: 'POST', headers: { 'x-maxy-key': KEY,
'content-type': 'application/json' }, body: JSON.stringify({ order: quote.order, signatures })
});
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/orders", headers={"x-maxy-key": KEY}, json={"order":
quote["order"], "signatures": signatures}).json()
```

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "expiresAt": 1794542691,
  "transaction": {
    "chainId": 8453,
    "from": "0xdd0d268Abc55f328F88f1Ee2e2f0850bFf43686b",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": {
    "ok": true
  },
  "next": {
    "method": "POST",
    "path": "/orders/submit",
    "body": {
      "order": "…",
      "signatures": "…",
      "txHash": "<transaction.simulatedHash>"
    }
  },
  "expiresAtIso": "2026-11-13T04:04:51.000Z",
  "refundAt": 1794542691,
  "sandbox": true
}
```


`POST /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` e6836a36-5e96-48f0-a94d-36486fe8db98), with long fields folded.

Fields: `orderId`, `route`, `submitted`, `expiresAt`, `expiresAtIso`, `refundAt`, `transaction`, `simulation`, `next`, `status`, `steps`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 409: `quote_expired`, `quote_mismatch`, `conflict`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2m POST /orders/submit

Tell Maxy the escrow or bridge transaction is sent (the solver also watches the chain). Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/submit" -H "content-type:
application/json" \
  -d "$(jq -c --arg sig "$SIG" --arg h "$TX_HASH" '{order: .order, signatures: {intent: $sig},
txHash: $h}' quote.json)"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/submit`, { method: 'POST', headers: { 'x-maxy-key': KEY,
'content-type': 'application/json' }, body: JSON.stringify({ ...placed.next.body, txHash }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/orders/submit", headers={"x-maxy-key": KEY}, json=
{**placed["next"]["body"], "txHash": tx_hash}).json()
```

Response

```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "notified": true,
  "status": "/status?
orderId=0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3&chainId=8453&route=evm-
btc-escrow",
  "sandbox": true
}
```

`POST /orders/submit`: a real sandbox answer, 13 November 2026 (`x-request-id` 2e7f1e4c-d9f9-4350-9ad5-13c93f7acf8a).

Fields: `orderId`, `route`, `notified`, `onChain`, `status`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2n GET /status

Where an order is. Needs your key.

Parameter	Required	Meaning
`orderId`	no	The order id; or give txHash.
`txHash`	no	Any transaction of the order: the escrow or settlement transaction, or the Bitcoin payment.
`chainId`	no	
`route`	no	
`toChainId`	no	cross-evm only.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/status?orderId=$ORDER_ID&chainId=8453"
```

TypeScript

```
const res = await fetch(`${BASE}/status?${new URLSearchParams({ orderId: orderId, chainId:
'8453' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/status", headers={"x-maxy-key": KEY}, params={"orderId":
order_id, "chainId": "8453"}).json()
```

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9",
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13"
    },
    {
      "label": "btc_payout",
      "hash": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success"
}
```


`GET /status`: a real sandbox answer, 13 November 2026 (`x-request-id` e828f87c-73d8-4eee-8ff3-ea5539d44411).

Fields: `orderId`, `route`, `state`, `rawStatus`, `isComplete`, `btcTxId`, `txs`, `refundAt`, `refundAtIso`, `refundAfterBlock`, `externalId`, `externalCustomerId`, `failureReason`, `scenario`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2o GET /orders/{id}

Where an order is (the same object as /status). Needs your key.

Parameter	Required	Meaning
`chainId`	no	
`route`	no	

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders/{order_id}", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9",
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13"
    },
    {
      "label": "btc_payout",
      "hash": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}` : a real sandbox answer, 13 November 2026 (`x-request-id` 93bd9560-1299-4260-bc3b-351e9c871a10).

Fields: `orderId`, `route`, `state`, `rawStatus`, `isComplete`, `btcTxId`, `txs`, `refundAt`, `refundAtIso`, `refundAfterBlock`, `externalId`, `externalCustomerId`, `failureReason`, `scenario`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`,

`tx_reverted`; 429: `rate_limited`. See error codes.

10.2p GET /orders/{id}/refund

The transaction that takes an expired escrow back to the user. Needs your key.

Parameter	Required	Meaning
`chainId`	yes	
`from`	no	Simulate from this sender. Default: the paying wallet.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/refund?chainId=8453"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/refund?${new URLSearchParams({ chainId:
'8453' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders/{order_id}/refund", headers={"x-maxy-key": KEY}, params=
{"chainId": "8453"}).json()
```

Response


```
{
  "orderId": "0x53414e44424f58ec837ff3f2e3b5ff0db94324f8ff30f5e2d799de1f997273cd",
  "refundable": true,
  "function": "cancelBuyEscrow",
  "explanation": "No solver committed before the escrow deadline: the escrow goes back to the
user. Sandbox: nothing to send; the order reads refunded shortly.",
  "transaction": {
    "chainId": 8453,
    "from": "0x30A7cba29a6f3C34c812fc64586f61A5872251A4",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f58ab2ab7c49ce88c7bddd2bb3fc3d3900a32641cf1bd7a1372be",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": {
    "ok": true
  },
  "sandbox": true
}
```


`GET /orders/{id}/refund` : a real sandbox answer, 13 November 2026 (`x-request-id` a84971e0-80f7-4991-8f4d-01d03dcf56d9).

Fields: `orderId`, `refundable`, `function`, `explanation`, `reason`, `unlocksAt`, `transaction`, `simulation`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2q GET /orders/{id}/deposit

Selling BTC: where and how much to pay, read from the settler. Needs your key.

Parameter	Required	Meaning
`chainId`	yes	
`maker`	no	

Parameter	Required	Meaning
`token`	no	
`sats`	no	
`minPayout`	no	
`lockBy`	no	

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/deposit?chainId=8453&sats=50000"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/deposit?${new URLSearchParams({ chainId: '8453', sats: '50000' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders/{order_id}/deposit", headers={"x-maxy-key": KEY}, params={"chainId": "8453", "sats": "50000"}).json()
```

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "chainId": 8453,
  "settler": "0x0000000000000000000000000000000000000000",
  "state": "COMMITTED",
  "ready": true,
  "address": "tb1qwrdwhtmx7snnesh4k9vt82ggn9xmyg8zwetx06",
  "sats": 50000,
  "amountBtc": "0.0005",
  "bitcoinUri": "bitcoin:tb1qwrdwhtmx7snnesh4k9vt82ggn9xmyg8zwetx06?amount=0.0005",
  "scriptPubKey": "0x001470daebaf66f4273cc2f5b158b3a908994db220e2",
  "payout": {
    "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
    "amount": "35320482",
    "to": "0x11B172F50AaD19f0372c3de1d2850cA1Cd180064"
  },
  "expiresAt": 1794539231,
  "btcDeadline": 0,
  "blocksLeft": 6,
  "verifierHeight": 0,
  "requiredConfirmations": 1,
  "intentHash": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "checked": "order",
  "simulated": true,
  "instructions": "Sandbox: do not pay. This is a testnet address, and the order moves on by
itself.",
  "verify": "…",
  "claim": "/orders/0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926/claim?
chainId=8453",
  "status": "/status?
orderId=0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926&chainId=8453&route=btc-
evm",
  "sandbox": true
}
```


`GET /orders/{id}/deposit` : a real sandbox answer, 13 November 2026 (`x-request-id` 58d489ce-b64a-4f0c-9220-6e350066ada1), with long fields folded.

Fields: `orderId`, `route`, `chainId`, `settler`, `state`, `ready`, `reason`, `address`, `satAmount`, `amountBtc`, `bitcoinUri`, `btcDeadline`, `retryAfterSeconds`, `lockBy`, `sats`, `scriptPubKey`, `payout`, `expiresAt`, `blocksLeft`, `verifierHeight`, `requiredConfirmations`, `intentHash`, `checked`, `simulated`, `instructions`, `verify`, `claim`, `status`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 409: `quote_expired`, `quote_mismatch`, `conflict`; 422: `amount_out_of_range`,

`order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2r GET /orders/{id}/claim

Selling BTC: the transaction that proves the payment and releases the payout to the user. Needs your key.

Parameter	Required	Meaning
`chainId`	yes	
`btcTxid`	no	
`from`	no	Simulate from this sender. Default: the wallet being paid.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/claim?chainId=8453"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/claim?${new URLSearchParams({ chainId: '8453' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders/{order_id}/claim", headers={"x-maxy-key": KEY}, params={"chainId": "8453"}).json()
```

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "chainId": 8453,
  "claimable": true,
  "function": "fillIntentWithFeeToken",
  "explanation": "Proves the BTC payment to the settler, which releases the payout to the maker.
Sandbox: nothing to send; the order completes by itself.",
  "btcTxid": "53414e44424f58f9ac0e49ea608e743552bca7a46625f415623b2b7c2144d436",
  "btcBlockHeight": 0,
  "payout": {
    "token": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
    "amount": "35320482",
    "to": "0x11B172F50AaD19f0372c3de1d2850cA1Cd180064"
  },
  "transaction": {
    "chainId": 8453,
    "from": "0x11B172F50AaD19f0372c3de1d2850cA1Cd180064",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f58282aa26e0d84fb2cf2505b5e6edeafa1f43680b552e1e2ed45",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": {
    "ok": true
  },
  "sandbox": true
}
```


`GET /orders/{id}/claim`: a real sandbox answer, 13 November 2026 (`x-request-id` a7477538-3026-40e7-a475-3f633477e530).

Fields: `orderId`, `claimable`, `reason`, `transaction`, `retryAfterSeconds`, `route`, `chainId`, `function`, `explanation`, `btcTxid`, `btcBlockHeight`, `payout`, `simulation`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2s GET /orders/{id}/bridge

Selling BTC for a coin with no settler: the bridge transaction, after the settler has paid. Needs your key.

Parameter	Required	Meaning
`chainId`	yes	
`toChainId`	yes	
`toToken`	yes	
`toAddress`	no	

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/bridge?chainId=8453"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/bridge?${new URLSearchParams({ chainId:
'8453' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders/{order_id}/bridge", headers={"x-maxy-key": KEY}, params=
{"chainId": "8453"}).json()
```

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "chainId": 8453,
  "toChainId": null,
  "toToken": null,
  "state": "FILLED",
  "ready": true,
  "toAmount": "35320482",
  "toAmountMin": "35320482",
  "transaction": {
    "chainId": 8453,
    "from": "0x11B172F50AaD19f0372c3de1d2850cA1Cd180064",
    "to": "0x0000000000000000000000000000000000000000",
    "data": "0x4d4158592053414e44424f583a204e4f542041205245414c205452414e53414354494f4e",
    "value": "0",
    "gas": "1",
    "simulatedHash": "0x53414e44424f58267912b85fdcd4f4d54e5d553e9f4abe3008ed3b75bf3742be",
    "simulated": true,
    "sandbox": true,
    "notice": "Sandbox transaction: do not broadcast. It goes to the zero address, moves nothing
and cannot do anything."
  },
  "simulation": {
    "ok": true
  },
  "description": "Sandbox: do not send it.",
  "sandbox": true
}
```


`GET /orders/{id}/bridge`: a real sandbox answer, 13 November 2026 (`x-request-id` 644b6c45-2854-4ca2-bbfe-c0641d9354bd).

Fields: `orderId`, `ready`, `reason`, `provider`, `expiresAt`, `transaction`, `approval`, `route`, `chainId`, `toChainId`, `toToken`, `state`, `toAmount`, `toAmountMin`, `simulation`, `description`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2t GET /orders/{id}/steps

Solana: where the offer is and what is left to sign. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/steps?route=sol-btc"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/steps?${new URLSearchParams({ route: 'sol-btc' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders/{order_id}/steps", headers={"x-maxy-key": KEY}, params={"route": "sol-btc"}).json()
```

Response

```
{
  "orderId": "53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db",
  "route": "sol-btc",
  "phase": "SANDBOX_PENDING",
  "state": "pending",
  "status": "/status?
orderId=53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db&route=sol-btc",
  "steps": [],
  "sandbox": true
}
```

`GET /orders/{id}/steps`: a real sandbox answer, 13 November 2026 (`x-request-id` cf12969c-6553-419f-815d-3b31728af867).

Fields: `orderId`, `route`, `status`, `phase`, `state`, `steps`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2u POST /orders/{id}/steps

Solana: the next stage, or a signed transaction to send. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/steps" -H "content-type: application/json" \
  -d "$(jq -c --arg tx "$SIGNED_TX" '.next.body + {transaction: $tx}' placed.json)"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/steps`, { method: 'POST', headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' }, body: JSON.stringify({ ...placed.next.body, transaction: signedTx }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/orders/{order_id}/steps", headers={"x-maxy-key": KEY}, json={**placed["next"]["body"], "transaction": signed_tx}).json()
```

Response


```
{
  "orderId": "53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db",
  "route": "sol-btc",
  "submitted": true,
  "signature":
"2wkCi1pKYrWwdYbupVbe7dNZNRYvXefjqot5kMjfyvxZu8wsJ2pQ6KJQynUUpGgVeTz7ZyKnGxrXXvgu5vhF8nvC",
  "txHashes": [
    "2wkCi1pKYrWwdYbupVbe7dNZNRYvXefjqot5kMjfyvxZu8wsJ2pQ6KJQynUUpGgVeTz7ZyKnGxrXXvgu5vhF8nvC"
  ],
  "simulated": true,
  "expiresAt": 1794535896,
  "status": "/status?
orderId=53414e44424f58cccc3b43bdecafae6d2ac0302b8ff9c504ab63dccd01d9a5db&route=sol-btc",
  "sandbox": true
}
```


`POST /orders/{id}/steps`: a real sandbox answer, 13 November 2026 (`x-request-id` d28192da-1864-4588-90cc-7013f4717174).

Fields: `orderId`, `route`, `status`, `phase`, `state`, `steps`, `done`, `submitted`, `signature`, `txHashes`, `simulated`, `expiresAt`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2v GET /orders

Your orders, newest first: placed through this API or the SDK with your key, or followed with /orders/watch. Needs your key.

Parameter	Required	Meaning
`from`	no	
`to`	no	
`format`	no	
`limit`	no	A page, newest first. A CSV export takes up to 1,000.
`cursor`	no	nextCursor of the page before.
`state`	no	
`wallet`	no	The paying wallet.
`externalId`	no	
`externalCustomerId`	no	

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders?from=2026-11-01&to=2026-12-01"
```

TypeScript

```
const res = await fetch(`${BASE}/orders?${new URLSearchParams({ from: '2026-11-01', to: '2026-12-01' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders", headers={"x-maxy-key": KEY}, params={"from": "2026-11-01", "to": "2026-12-01"}).json()
```

Response


```
{
  "orders": [
    {
      "orderId": "0x53414e44424f58bd874a9c3b8b3e2b671f7f15d92a42a9c9c34bd5d8079ca56b",
      "route": "evm-btc-gasless",
      "chainId": 8453,
      "state": "completed",
      "btcTxId": "53414e44424f58ce5c8ed9d6d714d8b1008a4797d5744581ac6f2d6193f13dfc",
      "createdAt": "2026-11-13T02:08:08.725062+00:00",
      "doneAt": "2026-11-13T02:09:01.000Z",
      "externalId": null,
      "externalCustomerId": null,
      "wallet": "0x174EeB02AC30add92d28f86401E5141cf3f9e480"
    },
    {
      "orderId": "0x53414e44424f5861c0d4304e38a2ea470fa92d4b3e57d4190e3d938018b72480",
      "route": "cross-evm",
      "chainId": 8453,
      "state": "completed",
      "btcTxId": null,
      "createdAt": "2026-11-13T02:08:03.886737+00:00",
      "doneAt": "2026-11-13T02:09:01.000Z",
      "externalId": null,
      "externalCustomerId": null,
      "wallet": "0x9b37eeb80632945471951d20610C45355Cdba69e"
    },
    "…"
  ],
  "nextCursor": null,
  "sandbox": true
}
```


`GET /orders`: a real sandbox answer, 13 November 2026 (`x-request-id` 31abf078-0414-42ac-9ba3-a91ca2d80f13), with long fields folded.

Fields: `orders`, `nextCursor`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2w POST /orders/watch

Follow any order for your webhooks. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/watch" -H "content-type:
application/json" \
  -d '{ "orderId": "'"$ORDER_ID"'", "chainId": 8453 }'
```

TypeScript

```
const res = await fetch(`${BASE}/orders/watch`, { method: 'POST', headers: { 'x-maxy-key': KEY,
'content-type': 'application/json' }, body: JSON.stringify({ orderId, chainId: 8453 }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/orders/watch", headers={"x-maxy-key": KEY}, json={"orderId":
order_id, "chainId": 8453}).json()
```

Response

```
{
  "watching": true,
  "orderId": "0x53414e44424f58bd874a9c3b8b3e2b671f7f15d92a42a9c9c34bd5d8079ca56b",
  "route": "evm-btc-gasless",
  "sandbox": true
}
```

`POST /orders/watch`: a real sandbox answer, 13 November 2026 (`x-request-id` 92ee587f-daf9-4e00-823d-10816b155d68).

Fields: `watching`, `orderId`, `route`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2x POST /webhooks

Register an HTTPS endpoint for signed order.updated events. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/webhooks" -H "content-type:
application/json" \
  -d '{ "url": "https://your.app/maxy/webhooks" }'
```

TypeScript

```
const res = await fetch(`${BASE}/webhooks`, { method: 'POST', headers: { 'x-maxy-key': KEY,
'content-type': 'application/json' }, body: JSON.stringify({ url:
'https://your.app/maxy/webhooks' }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/webhooks", headers={"x-maxy-key": KEY}, json={"url":
"https://your.app/maxy/webhooks"}).json()
```

Response

```
{
  "id": "7d320755-9eac-435b-9b4b-d8151856cdde",
  "url": "https://postman-echo.com/post",
  "events": [
    "order.updated"
  ],
  "secret": "whsec_… (shown once: store it)",
  "createdAt": "2026-11-13T02:08:05.553716+00:00",
  "sandbox": true
}
```

`POST /webhooks` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` c93801d0-4315-40ac-8f99-cac312d79550).

Fields: `id`, `url`, `secret`, `events`, `createdAt`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 409: `quote_expired`, `quote_mismatch`, `conflict`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2y GET /webhooks

Your endpoints (without secrets). Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/webhooks"
```

TypeScript

```
const res = await fetch(`${BASE}/webhooks`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/webhooks", headers={"x-maxy-key": KEY}).json()
```

Response

```
{
  "webhooks": [
    {
      "id": "7d320755-9eac-435b-9b4b-d8151856cdde",
      "url": "https://postman-echo.com/post",
      "events": [
        "order.updated"
      ],
      "createdAt": "2026-11-13T02:08:05.553716+00:00"
    }
  ],
  "sandbox": true
}
```

`GET /webhooks`: a real sandbox answer, 13 November 2026 (`x-request-id` 92b75e93-6cea-4225-8380-e9438ce1fa54).

Fields: `webhooks`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2z DELETE /webhooks/{id}

Stop sending to an endpoint. Needs your key.

curl

```
curl -s -X DELETE -H "x-maxy-key: $MAXY_API_KEY" "$BASE/webhooks/$WEBHOOK_ID"
```

TypeScript

```
const res = await fetch(`${BASE}/webhooks/${webhookId}`, { method: 'DELETE', headers: { 'x-maxy-
key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.delete(f"{BASE}/webhooks/{webhook_id}", headers={"x-maxy-key": KEY}).json()
```

Response

```
{
  "deleted": true,
  "id": "7d320755-9eac-435b-9b4b-d8151856cdde",
  "sandbox": true
}
```

`DELETE /webhooks/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 625f4217-03fa-41cf-8862-5ac41bc920b2).

Fields: `deleted`, `id`.

Errors: 404: `not_found`. See error codes.

10.2aa POST /webhooks/{id}/test

Send a signed webhook.test event now. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/webhooks/$WEBHOOK_ID/test"
```

TypeScript

```
const res = await fetch(`${BASE}/webhooks/${webhookId}/test`, { method: 'POST', headers: { 'x-
maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/webhooks/{webhook_id}/test", headers={"x-maxy-key": KEY}).json()
```

Response

```
{
  "eventId": "evt_04417bf111cda494f5b148a3",
  "delivered": true,
  "status": 200,
  "error": null,
  "sandbox": true
}
```

`POST /webhooks/{id}/test`: a real sandbox answer, 13 November 2026 (`x-request-id` 7e45d64f-46cc-4eaf-b4a9-d07f7c5d371b).

Fields: `delivered`, `status`, `eventId`, `error`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2ab GET /webhooks/{id}/events

The last 25 events of an endpoint, with attempts and answers. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/webhooks/$WEBHOOK_ID/events"
```

TypeScript

```
const res = await fetch(`${BASE}/webhooks/${webhookId}/events`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/webhooks/{webhook_id}/events", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "events": [
    {
      "id": "evt_0d4b0ca996c26c37e45f72c6_replay",
      "type": "order.updated",
      "createdAt": "2026-11-13T02:08:30.84+00:00",
      "attempts": 0,
      "deliveredAt": null,
      "failedAt": null,
      "lastStatus": null,
      "lastError": null,
      "data": {
        "txs": [],
        "route": "evm-btc-gasless",
        "state": "in_progress",
        "btcTxId": null,
        "chainId": 8453,
        "orderId": "0x53414e44424f58bd874a9c3b8b3e2b671f7f15d92a42a9c9c34bd5d8079ca56b",
        "sandbox": true,
        "scenario": "webhook_replay",
        "rawStatus": "SANDBOX_FILLING",
        "isComplete": false,
        "previousState": "pending"
      }
    },
    "…"
  ],
  "sandbox": true
}
```


`GET /webhooks/{id}/events`: a real sandbox answer, 13 November 2026 (`x-request-id` bff7e43b-d7ba-4e1b-9560-6f372d7b8f8d), with long fields folded.

Fields: `events`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2ac GET /sandbox/scenarios

Sandbox only: the scenarios you can force. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/sandbox/scenarios"
```

TypeScript

```
const res = await fetch(`${BASE}/sandbox/scenarios`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/sandbox/scenarios", headers={"x-maxy-key": KEY}).json()
```

Response

```
{
  "scenarios": [
    {
      "name": "success",
      "what": "The default. The order walks pending, in_progress, completed, with a sandbox
btcTxId."
    },
    {
      "name": "refund_available",
      "what": "The order reaches refundable. GET /orders/{id}/refund then answers a refund; the
order reads refunded 15 s after you ask."
    },
    "…"
  ],
  "sandbox": true
}
```

`GET /sandbox/scenarios`: a real sandbox answer, 13 November 2026 (`x-request-id` c628ef67-3029-4b07-b3fe-78be13aca7b4), with long fields folded.

Fields: `scenarios`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ad GET /sandbox/rates

Sandbox only: live BTC, ETH and USDC prices in dollars. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/sandbox/rates"
```

TypeScript

```
const res = await fetch(`${BASE}/sandbox/rates`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/sandbox/rates", headers={"x-maxy-key": KEY}).json()
```

Response

```
{
  "btcUsd": 70782.2587885,
  "ethUsd": 2007.456865575,
  "usdcUsd": 1,
  "sandbox": true
}
```

`GET /sandbox/rates`: a real sandbox answer, 13 November 2026 (`x-request-id` 846b00f2-f5c6-498f-85fa-09b2b282d412).

Fields: `btcUsd`, `ethUsd`, `usdcUsd`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ae POST /quotes/batch

Up to 20 prices or quotes in one call, with an answer or an error for each. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/quotes/batch" -H "content-type:
application/json" \
  -d '{ "requests": [{ "from": "ETH on base", "to": "BTC", "amount": "0.004" }] }'
```

TypeScript

```
const res = await fetch(`${BASE}/quotes/batch`, { method: 'POST', headers: { 'x-maxy-key': KEY,
'content-type': 'application/json' }, body: JSON.stringify({ requests: [{ from: 'ETH on base',
to: 'BTC', amount: '0.004' }] }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/quotes/batch", headers={"x-maxy-key": KEY}, json={"requests":
[{"from": "ETH on base", "to": "BTC", "amount": "0.004"}]}).json()
```

Response


```
{
  "results": [
    {
      "index": 0,
      "status": 200,
      "requestId": "0cf5d4fd-1d71-461e-ac64-9bbba4d323d8",
      "quote": "…"
    },
    {
      "index": 1,
      "status": 200,
      "requestId": "ac3faae4-3d5c-42f1-9a43-12305c32c7df",
      "quote": "…"
    },
    {
      "index": 2,
      "status": 400,
      "requestId": "73b1d8bf-c950-4f78-ab64-38397ffc7e85",
      "error": {
        "code": "invalid_input",
        "message": "PEPE is not a coin Maxy knows on Base (it knows ETH, USDC, USDT, WBTC,
cbBTC, MORPHO, USDe). Pass its contract address instead.",
        "retryable": false,
        "docs": "https://maxy.co/docs/sdk-errors#invalid_input"
      }
    }
  ]
}
```


`POST /quotes/batch`: a real sandbox answer, 13 November 2026 (`x-request-id` b2cf35b7-32eb-4af2-9a91-95355871ece8), with long fields folded.

Fields: `results`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2af GET /prices/stream

Live prices as server-sent events: the same question as GET /price. Needs your key.

Parameter	Required	Meaning
`from`	no	What you sell, in words: "ETH", "USDC on arbitrum", "BASE.ETH", "BTC", "SOL", "USDC on eip155:8453". Or use fromChain + fromToken.
`to`	no	What you buy, in words. A CAIP-2 id alone is its chain's own coin (bip122:000000000019d6689c085ae165831e93 is BTC). Or use toChain + toToken.
`amount`	no	How much you sell, in whole coins. Or use fromAmount.
`fromChain`	no	Chain id, name or CAIP-2 id (8453, base, eip155:8453, bitcoin, solana). Alias: chainId.
`fromToken`	no	Token address or symbol. Alias: sellToken.
`toChain`	no	Chain id, name or CAIP-2 id; defaults to bitcoin when toToken is BTC.
`toToken`	no	Token address or symbol. Alias: buyToken.
`fromAmount`	no	How much you sell, in the smallest unit. Alias: sellAmount.
`scenario`	no	Sandbox only: forces an outcome with a test key (GET /sandbox/scenarios lists them). A live key ignores it.
`simulate`	no	Sandbox only: the same as scenario, also by other names: quote_expired, fill_timeout, underpaid, refund.
`intervalSeconds`	no	
`ticket`	no	In place of the key, from a browser: the ticket from the stream-ticket route.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/prices/stream?
from=ETH%20on%20base&to=BTC&amount=0.004"
```

TypeScript

```
const res = await fetch(`${BASE}/prices/stream?${new URLSearchParams({ from: 'ETH on base', to:
'BTC', amount: '0.004' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/prices/stream", headers={"x-maxy-key": KEY}, params={"from": "ETH
on base", "to": "BTC", "amount": "0.004"}).json()
```

```
retry: 5000

event: price
data: {"route":"evm-btc-escrow","liquidityAvailable":true,"from":
{"chain":8453,"chainName":"Base","token":"0x0000000000000000000000000000000000000000","symbol":"ETH
{"chain":"bitcoin","chainName":"Bitcoin","token":"BTC","symbol":"BTC","decimals":8,"caip2":"bip122:
{"send":"0.004 ETH on Base","receive":"0.00010868 BTC","pair":"ETH on Base →
BTC"},"expiresAt":1794536203,"expiresAtIso":"2026-11-13T02:16:43.000Z","limits":
{"minSats":546},"sandbox":true}
```


`GET /prices/stream`: a real sandbox answer, 13 November 2026 (`x-request-id` 744113cb-861c-477c-940d-adb3be1a2cd2).

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ag POST /prices/stream-ticket

A ticket for GET /prices/stream from a browser, for this pair. Needs your key.

Parameter	Required	Meaning
`from`	no	What you sell, in words: "ETH", "USDC on arbitrum", "BASE.ETH", "BTC", "SOL", "USDC on eip155:8453". Or use fromChain + fromToken.
`to`	no	What you buy, in words. A CAIP-2 id alone is its chain's own coin (bip122:000000000019d6689c085ae165831e93 is BTC). Or use toChain + toToken.
`amount`	no	How much you sell, in whole coins. Or use fromAmount.
`fromChain`	no	Chain id, name or CAIP-2 id (8453, base, eip155:8453, bitcoin, solana). Alias: chainId.
`fromToken`	no	Token address or symbol. Alias: sellToken.
`toChain`	no	Chain id, name or CAIP-2 id; defaults to bitcoin when toToken is BTC.
`toToken`	no	Token address or symbol. Alias: buyToken.
`fromAmount`	no	How much you sell, in the smallest unit. Alias: sellAmount.
`scenario`	no	Sandbox only: forces an outcome with a test key (GET /sandbox/scenarios lists them). A live key ignores it.
`simulate`	no	Sandbox only: the same as scenario, also by other names: quote_expired, fill_timeout, underpaid, refund.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/prices/stream-ticket?
from=ETH%20on%20base&to=BTC"
```

TypeScript

```
const res = await fetch(`${BASE}/prices/stream-ticket?${new URLSearchParams({ from: 'ETH on
base', to: 'BTC' })}`, { method: 'POST', headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/prices/stream-ticket", headers={"x-maxy-key": KEY}, params=
{"from": "ETH on base", "to": "BTC"}).json()
```

Response

```
{
  "ticket":
"mst1.eyJrIjoicCIsInQiOiJmcm9tPWV0aCBvbiBiYXNlJmZyb21DaGFpbj0mZnJvbVRva2VuPSZ0bz1idGMmdG9DaGFpbj0md
  "expiresAt": 1794536094,
  "expiresAtIso": "2026-11-13T02:14:54.000Z",
  "stream": "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api/prices/stream?
from=ETH+on+base&to=BTC&ticket=mst1.eyJrIjoicCIsInQiOiJmcm9tPWV0aCBvbiBiYXNlJmZyb21DaGFpbj0mZnJvbVR
  "sandbox": true
}
```

`POST /prices/stream-ticket` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` 9b12e15b-b00e-4e5b-8cc2-68cc141f87d2).

Fields: `ticket`, `expiresAt`, `expiresAtIso`, `stream`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ah POST /orders/{id}/signature

The same as POST /orders, for the order in the path. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/signature" -H "content-
type: application/json" \
  -d "$(jq -c --arg sig "$SIG" '{order: .order, signature: $sig}' quote.json)"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/signature`, { method: 'POST', headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' }, body: JSON.stringify({ order: quote.order, signature }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/orders/{order_id}/signature", headers={"x-maxy-key": KEY}, json={"order": quote["order"], "signature": signature}).json()
```

Response

```
{
  "orderId": "0x53414e44424f586554db1ea061d2a85e6cd342692eb7c952f7e95ac9b8415022",
  "route": "evm-btc-escrow",
  "expiresAt": 1794542953,
  "transaction": "…",
  "simulation": "…",
  "next": "…",
  "expiresAtIso": "2026-11-13T04:09:13.000Z",
  "refundAt": 1794542953,
  "sandbox": true
}
```

`POST /orders/{id}/signature` : a real sandbox answer, 13 November 2026 (`x-request-id` b087df85-664a-494a-92c2-c799389b2a53), with long fields folded.

Fields: `orderId` , `route` , `submitted` , `expiresAt` , `expiresAtIso` , `refundAt` , `transaction` , `simulation` , `next` , `status` , `steps` .

Errors: 400: `invalid_input` ; 401: `auth_required` , `invalid_api_key` ; 403: `key_revoked` ; 422: `amount_out_of_range` , `order_cap_exceeded` , `insufficient_funds` , `insufficient_liquidity` , `tx_reverted` ; 429: `rate_limited` . See error codes.

10.2ai POST /orders/{id}/cancel

The transaction that takes back an escrow no solver committed to. Needs your key.

Parameter	Required	Meaning
`chainId`	yes	
`from`	no	The wallet that will send it; defaults to the escrow's maker.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/cancel?chainId=8453"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/cancel?${new URLSearchParams({ chainId:
'8453' })}`, { method: 'POST', headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/orders/{order_id}/cancel", headers={"x-maxy-key": KEY}, params=
{"chainId": "8453"}).json()
```

Response


```
{
  "orderId": "0x53414e44424f58d83c861d140370c484eb094547ddc05b8e7ac093b585326213",
  "state": "pending",
  "cancellable": true,
  "function": "cancelBuyEscrow",
  "allowedStates": [
    "pending"
  ],
  "explanation": "Takes back the escrow: no solver has committed yet. Sandbox: nothing to send;
the order now reads cancelled.",
  "transaction": "…",
  "sandbox": true
}
```


`POST /orders/{id}/cancel`: a real sandbox answer, 13 November 2026 (`x-request-id` 3b4d6287-ca5d-4059-a2cc-e952d9efaf20), with long fields folded.

Fields: `orderId`, `cancellable`, `function`, `explanation`, `reason`, `transaction`, `simulation`, `state`, `allowedStates`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 409: `quote_expired`, `quote_mismatch`, `conflict`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2aj GET /orders/{id}/events

The order's states so far, with times. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/events"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/events`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders/{order_id}/events", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "orderId": "0x53414e44424f5823936eadc65c68429f1fc465d671b4382365c5ff83ffbf13ae",
  "events": [
    {
      "state": "pending",
      "previousState": null,
      "rawStatus": "SANDBOX_PENDING",
      "btcTxId": null,
      "txs": [],
      "at": "2026-11-13T02:09:03.000Z"
    },
    {
      "state": "in_progress",
      "previousState": "pending",
      "rawStatus": "SANDBOX_FILLING",
      "btcTxId": null,
      "txs": [],
      "at": "2026-11-13T02:08:54.000Z"
    },
    {
      "state": "completed",
      "previousState": "in_progress",
      "rawStatus": "SANDBOX_COMPLETED",
      "btcTxId": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7",
      "txs": [
        {
          "label": "btc_payout",
          "hash": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7"
        }
      ],
      "at": "2026-11-13T02:09:06.000Z"
    }
  ],
  "nextCursor": null,
  "sandbox": true
}
```


`GET /orders/{id}/events`: a real sandbox answer, 13 November 2026 (`x-request-id` 38d7829b-fba3-4d13-9e17-26e684d7594d).

Fields: `orderId`, `events`, `nextCursor`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ak GET /orders/{id}/events/stream

The order's state as server-sent events, until it ends. Needs your key.

Parameter	Required	Meaning
`chainId`	no	
`route`	no	
`ticket`	no	In place of the key, from a browser: the ticket from the stream-ticket route.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/events/stream?chainId=8453"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/events/stream?${new URLSearchParams({
chainId: '8453' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/orders/{order_id}/events/stream", headers={"x-maxy-key": KEY},
params={"chainId": "8453"}).json()
```

```
retry: 5000

event: price
data: {"route":"evm-btc-escrow","liquidityAvailable":true,"from":
{"chain":8453,"chainName":"Base","token":"0x0000000000000000000000000000000000000000","symbol":"ETH
{"chain":"bitcoin","chainName":"Bitcoin","token":"BTC","symbol":"BTC","decimals":8,"caip2":"bip122:
{"send":"0.004 ETH on Base","receive":"0.00010868 BTC","pair":"ETH on Base →
BTC"},"expiresAt":1794536203,"expiresAtIso":"2026-11-13T02:16:43.000Z","limits":
{"minSats":546},"sandbox":true}
```

`GET /prices/stream`: a real sandbox answer, 13 November 2026 (`x-request-id` 744113cb-861c-477c-940d-adb3be1a2cd2).

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2al POST /orders/{id}/stream-ticket

A ticket for GET /orders/{id}/events/stream from a browser, for this order. Needs your key.

Parameter	Required	Meaning
`chainId`	no	
`route`	no	

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/orders/$ORDER_ID/stream-ticket?
chainId=8453"
```

TypeScript

```
const res = await fetch(`${BASE}/orders/${orderId}/stream-ticket?${new URLSearchParams({
chainId: '8453' })}`, { method: 'POST', headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/orders/{order_id}/stream-ticket", headers={"x-maxy-key": KEY},
params={"chainId": "8453"}).json()
```

Response


```
{
  "ticket":
"mst1.eyJrIjoibyIsInQiOiIweDUzNDE0ZTQ0NDI0ZjU4OWFhNjRhOGMwZWU2ZTFjYzZjOTZmNDU0OWE0NmY4ZGJjYmFlZjI3M
  "expiresAt": 1794536094,
  "expiresAtIso": "2026-11-13T02:14:54.000Z",
  "stream": "https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-
api/orders/0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3/events/stream?
chainId=8453&ticket=mst1.eyJrIjoibyIsInQiOiIweDUzNDE0ZTQ0NDI0ZjU4OWFhNjRhOGMwZWU2ZTFjYzZjOTZmNDU0OW
  "sandbox": true
}
```


`POST /orders/{id}/stream-ticket` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` b7e4e48b-4d58-4258-b02e-1c4a24b02995).

Fields: `ticket`, `expiresAt`, `expiresAtIso`, `stream`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2am POST /webhooks/{id}/events/{eventId}/replay

Send a past event again now, with a new signature and the same event id. Needs your key.

Parameter	Required	Meaning
`eventId`	yes	

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY"
"$BASE/webhooks/$WEBHOOK_ID/events/$EVENT_ID/replay"
```

TypeScript

```
const res = await fetch(`${BASE}/webhooks/${webhookId}/events/${eventId}/replay`, { method:
'POST', headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/webhooks/{webhook_id}/events/{event_id}/replay", headers={"x-
maxy-key": KEY}).json()
```

Response

```
{
  "eventId": "evt_0d4b0ca996c26c37e45f72c6_replay",
  "replayed": true,
  "delivered": true,
  "status": 200,
  "error": null,
  "sandbox": true
}
```

`POST /webhooks/{id}/events/evt_0d4b0ca996c26c37e45f72c6_replay/replay`: a real sandbox answer, 13 November 2026 (`x-request-id` 7409d1ad-ff8e-406c-8500-15720f1915ec).

Fields: `replayed`, `eventId`, `delivered`, `status`, `error`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2an POST /sandbox/keys

A test key and a test publishable key, with no sign-up. No key needed.

curl

```
curl -s -X POST "$BASE/sandbox/keys" -H "content-type: application/json" \
  -d '{ "name": "my project", "origins": ["http://localhost:3000"] }'
```

TypeScript

```
const res = await fetch(`${BASE}/sandbox/keys`, { method: 'POST', headers: { 'content-type':
'application/json' }, body: JSON.stringify({ name: 'my project', origins:
['http://localhost:3000'] }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/sandbox/keys", json={"name": "my project", "origins":
["http://localhost:3000"]}).json()
```

Response

```
{
  "secretKey": "mxk_test_fBFzTP…",
  "publishableKey": "mxk_pk_test_ZnoGp6…",
  "origins": [
    "http://localhost:3000"
  ],
  "orderCapUsd": 100,
  "note": "Test keys: the sandbox only, nothing real moves. Shown once; Maxy keeps only a hash.
Unused for 30 days, they are revoked."
}
```

`POST /sandbox/keys` (HTTP 201): a real sandbox answer, 5 November 2026 (`x-request-id` ee2e78cb-478b-4d65-8f53-8dcb63f4c103).

Fields: `secretKey`, `publishableKey`, `origins`, `orderCapUsd`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ao GET /account

Your account: keys, usage, limit requests. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/account"
```

TypeScript

```
const res = await fetch(`${BASE}/account`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/account", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "partner": "Self-serve test account",
  "selfServe": true,
  "environment": "test",
  "keys": [
    {
      "prefix": "mxk_pk_test_K7M5zQ",
      "kind": "publishable",
      "environment": "test",
      "origins": [
        "http://localhost:3000"
      ],
      "orderCapUsd": 100,
      "createdAt": "2026-11-13T00:54:24.702925+00:00",
      "revokedAt": null,
      "you": false
    },
    {
      "prefix": "mxk_test_8GOVco",
      "kind": "secret",
      "environment": "test",
      "origins": [],
      "orderCapUsd": 100,
      "createdAt": "2026-11-13T00:54:24.654013+00:00",
      "revokedAt": null,
      "you": true
    }
  ],
  "usage": [
    {
      "day": "2026-11-13",
      "calls": 525,
      "errors": 41
    }
  ],
  "limitRequests": []
}
```


`GET /account`: a real sandbox answer, 13 November 2026 (`x-request-id` dfab347c-c566-46cb-8113-5f8ce4395fbc), with long fields folded.

Fields: `partner`, `selfServe`, `environment`, `keys`, `usage`, `limitRequests`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ap GET /usage

Your calls per day. Needs your key.

Parameter	Required	Meaning
`days`	no	

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/usage?days=7"
```

TypeScript

```
const res = await fetch(`${BASE}/usage?${new URLSearchParams({ days: '7' })}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/usage", headers={"x-maxy-key": KEY}, params={"days": "7"}).json()
```

Response

```
{
  "days": 7,
  "usage": [
    {
      "day": "2026-11-13",
      "calls": 526,
      "errors": 41
    }
  ]
}
```

`GET /usage`: a real sandbox answer, 13 November 2026 (`x-request-id` 5e80340a-4c89-4aa1-99bc-e5509199bc26), with long fields folded.

Fields: `days`, `usage`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2aq POST /keys

A new publishable key, or with a test key another test key. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/keys" -H "content-type: application/json"
\
  -d '{ "kind": "publishable", "origins": ["https://app.example.com"] }'
```

TypeScript

```
const res = await fetch(`${BASE}/keys`, { method: 'POST', headers: { 'x-maxy-key': KEY,
'content-type': 'application/json' }, body: JSON.stringify({ kind: 'publishable', origins:
['https://app.example.com'] }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/keys", headers={"x-maxy-key": KEY}, json={"kind": "publishable",
"origins": ["https://app.example.com"]}).json()
```

Response

```
{
  "key": "mxk_pk_test_12cyLH…",
  "prefix": "mxk_pk_test_12cyLH",
  "kind": "publishable",
  "origins": [
    "https://app.example.com"
  ],
  "orderCapUsd": 100,
  "note": "Shown once. Maxy keeps only a hash."
}
```

`POST /keys` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` fc9cb30e-0303-44dc-b9e8-b954016f5389).

Fields: `key`, `prefix`, `kind`, `origins`, `orderCapUsd`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ar POST /keys/rotate

A new secret key with the same cap; the old one works until you revoke it. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/keys/rotate"
```

TypeScript

```
const res = await fetch(`${BASE}/keys/rotate`, { method: 'POST', headers: { 'x-maxy-key': KEY }
});
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/keys/rotate", headers={"x-maxy-key": KEY}).json()
```

Response

```
{
  "key": "mxk_test_jbTrfS…",
  "prefix": "mxk_test_jbTrfS",
  "rotatedFrom": "mxk_test_8GOVco",
  "orderCapUsd": 100,
  "note": "Shown once. mxk_test_8GOVco… keeps working until you revoke it: DELETE
/keys/mxk_test_8GOVco."
}
```

`POST /keys/rotate` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` a3662660-9cf5-4fca-a443-a17cdbd980bd).

Fields: `key`, `prefix`, `rotatedFrom`, `orderCapUsd`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2as POST /keys/{prefix}/origins

Change a publishable key's browser origins. Needs your key.

Parameter	Required	Meaning
`prefix`	yes	

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/keys/{prefix}/origins" -H "content-type: application/json" \
  -d '{ "origins": ["https://app.example.com"] }'
```

TypeScript

```
const res = await fetch(`${BASE}/keys/{prefix}/origins`, { method: 'POST', headers: { 'x-maxy-key': KEY, 'content-type': 'application/json' }, body: JSON.stringify({ origins: ['https://app.example.com'] }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/keys/{prefix}/origins", headers={"x-maxy-key": KEY}, json={"origins": ["https://app.example.com"]}).json()
```

Response

```
{
  "prefix": "mxk_pk_test_12cyLH",
  "origins": [
    "https://app.example.com",
    "https://www.example.com"
  ],
  "note": "Applies within a minute."
}
```

`POST /keys/mxk_pk_test_12cyLH/origins`: a real sandbox answer, 13 November 2026 (`x-request-id` 03c57405-9c71-485d-a0d0-ccae1c0e193b).

Fields: `prefix`, `origins`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2at DELETE /keys/{prefix}

Revoke a key of this account. Needs your key.

Parameter	Required	Meaning
`prefix`	yes	
`confirm`	no	Needed to revoke the key you are calling with.

curl

```
curl -s -X DELETE -H "x-maxy-key: $MAXY_API_KEY" "$BASE/keys/{prefix}"
```

TypeScript

```
const res = await fetch(`${BASE}/keys/{prefix}`, { method: 'DELETE', headers: { 'x-maxy-key':
KEY } });
const answer = await res.json();
```

Python

```
answer = requests.delete(f"{BASE}/keys/{prefix}", headers={"x-maxy-key": KEY}).json()
```

Response

```
{
  "revoked": true,
  "prefix": "mxk_test_jbTrfS",
  "note": "It stops working within a minute."
}
```

`DELETE /keys/mxk_test_jbTrfS`: a real sandbox answer, 13 November 2026 (`x-request-id` 2e8fbfd4-8736-4cbb-8741-37b986342119).

Fields: `revoked`, `prefix`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2au POST /limits/requests

Ask Maxy for a higher order size cap. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/limits/requests" -H "content-type:
application/json" \
  -d '{ "wantedUsd": 1000, "note": "launch" }'
```

TypeScript

```
const res = await fetch(`${BASE}/limits/requests`, { method: 'POST', headers: { 'x-maxy-key':
KEY, 'content-type': 'application/json' }, body: JSON.stringify({ wantedUsd: 1000, note:
'launch' }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/limits/requests", headers={"x-maxy-key": KEY}, json=
{"wantedUsd": 1000, "note": "launch"}).json()
```

Response

```
{
  "id": "8a01dbcd-258b-470a-9e92-22f0c91f4627",
  "createdAt": "2026-11-13T02:09:31.167606+00:00",
  "wantedUsd": 1000,
  "note": "A person at Maxy answers this. Your current cap stays until then."
}
```

`POST /limits/requests` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` 0f79c578-0a97-4508-8fcf-3c062c18185e).

Fields: `id`, `wantedUsd`, `note`, `createdAt`, `answeredAt`, `answer`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2av GET /limits/requests

What you asked for, and the answers. Needs your key.

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/limits/requests"
```

TypeScript

```
const res = await fetch(`${BASE}/limits/requests`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/limits/requests", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "requests": [
    {
      "id": "8a01dbcd-258b-470a-9e92-22f0c91f4627",
      "wantedUsd": 1000,
      "note": "Docs capture: please ignore.",
      "createdAt": "2026-11-13T02:09:31.167606+00:00",
      "answeredAt": null,
      "answer": null
    }
  ]
}
```


`GET /limits/requests`: a real sandbox answer, 13 November 2026 (`x-request-id` 3493f1d8-439d-48ba-8c17-1278d64241ca), with long fields folded.

Fields: `requests`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2aw GET /statements

A month of your orders with totals. Needs your key.

Parameter	Required	Meaning
`month`	no	
`format`	no	

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/statements?month=2026-11"
```

TypeScript

```
const res = await fetch(`${BASE}/statements?${new URLSearchParams({ month: '2026-11' })}`, {
headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/statements", headers={"x-maxy-key": KEY}, params={"month": "2026-11"}).json()
```

Response


```
{
  "partner": "Self-serve test account",
  "month": "2026-11",
  "sandbox": true,
  "generatedAt": "2026-11-13T02:09:25.916Z",
  "totals": {
    "orders": 48,
    "completed": 29,
    "refunded": 3,
    "open": 5,
    "failed": 11,
    "completedSats": 20892597,
    "byRoute": {
      "evm-btc-gasless": 20,
      "evm-btc-escrow": 15,
      "btc-evm": 5,
      "sol-btc": 3,
      "btc-sol": 3,
      "cross-evm": 2
    }
  },
  "orders": [
    {
      "orderId": "0x53414e44424f584384defa1bcd22cb5695ec6fb84e00b54e4bb867d6cc88679e",
      "route": "evm-btc-gasless",
      "chainId": 8453,
      "state": "completed",
      "btcTxId": "53414e44424f5827523f9fa7c9ab64b791b5991f7ed6e7b7ba13211dcb4750bf",
      "satAmount": 13598,
      "createdAt": "2026-11-13T00:55:13.585716+00:00",
      "doneAt": "2026-11-13T00:56:01.271+00:00",
      "externalId": null,
      "externalCustomerId": null,
      "wallet": "0xfeA476B9dc70a4dA9dbA3124209C0072e9F682EE"
    },
    {
      "orderId": "0x53414e44424f58a55336e8fd9fb67e3c34297c28ae97995091dca95cf9f0e9cc",
      "route": "evm-btc-gasless",
      "chainId": 8453,
      "state": "completed",
      "btcTxId": "53414e44424f58206efdc274b58f8f0c973c3e59b5d84cc8a7e7ca188503416c",
      "satAmount": 13584,
      "createdAt": "2026-11-13T00:57:41.546086+00:00",
      "doneAt": "2026-11-13T00:58:31+00:00",
      "externalId": null,
      "externalCustomerId": null,
      "wallet": "0xa6a0745Aaf9173f4b3D40f2aF5c97FA339dd11d2"
```

```
    },
    "…"
  ]
}
```

`GET /statements`: a real sandbox answer, 13 November 2026 (`x-request-id` eff7a5ef-d53f-4f28-a627-3bea09903cce), with long fields folded.

Fields: `partner`, `month`, `generatedAt`, `totals`, `orders`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ax GET /checkout/verify

Hosted checkout: is this signed URL what the partner signed. No key needed.

Parameter	Required	Meaning
`apiKey`	no	
`kid`	no	
`exp`	no	
`sig`	no	
`defaultAmount`	no	
`from`	no	
`recipient`	no	
`externalId`	no	
`externalCustomerId`	no	
`redirectUrl`	no	
`theme`	no	
`locale`	no	

curl

```
curl -s "$BASE/checkout/verify?apiKey=$MAXY_PUBLISHABLE_KEY&kid=$KEY_PREFIX&exp=$EXP&sig=$SIG"
```

TypeScript

```
const res = await fetch(`${BASE}/checkout/verify?${new URLSearchParams({ apiKey: undefined, kid: undefined, exp: undefined, sig: undefined })}`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/checkout/verify", params={"apiKey": undefined, "kid": undefined, "exp": undefined, "sig": undefined}).json()
```

Response

```
{
  "error": {
    "code": "checkout_signature_invalid",
    "message": "This checkout link was changed, or its key was revoked. Ask the site that sent you for a new one.",
    "retryable": false,
    "docs": "https://maxy.co/docs/sdk-errors#checkout_signature_invalid"
  },
  "requestId": "807ab484-e8c0-4931-ba40-b90c81b9e4a5"
}
```

`GET /checkout/verify` (HTTP 403): a real sandbox answer, 13 November 2026 (`x-request-id` 807ab484-e8c0-4931-ba40-b90c81b9e4a5).

Fields: `valid`, `apiKey`, `params`, `frameAncestors`, `expiresAt`, `error`, `requestId`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ay GET /explorer/{query}

Look up an order by order id, intent hash, chain transaction or Bitcoin txid. Needs your key.

Parameter	Required	Meaning
`query`	yes	

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/explorer/{query}"
```

TypeScript

```
const res = await fetch(`${BASE}/explorer/{query}`, { headers: { 'x-maxy-key': KEY } });
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/explorer/{query}", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "query": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "sandbox": true,
  "order": {
    "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
    "route": "evm-btc-escrow",
    "state": "completed",
    "rawStatus": "SANDBOX_COMPLETED",
    "isComplete": true,
    "btcTxId": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9",
    "txs": [
      {
        "label": "escrow",
        "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13"
      },
      {
        "label": "btc_payout",
        "hash": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9"
      }
    ],
    "failureReason": null,
    "sandbox": true,
    "scenario": "success"
  },
  "chainId": 8453,
  "timeline": [
    {
      "state": "pending",
      "rawStatus": "SANDBOX_PENDING",
      "at": "2026-11-13T02:04:51.000Z"
    },
    {
      "state": "in_progress",
      "rawStatus": "SANDBOX_FILLING",
      "at": "2026-11-13T02:05:00.000Z"
    },
    {
      "state": "completed",
      "rawStatus": "SANDBOX_COMPLETED",
      "at": "2026-11-13T02:05:12.000Z"
    }
  ],
  "links": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13",
      "url": null
```

```
    },
    {
      "label": "btc_payout",
      "hash": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9",
      "url": null
    }
  ]
}
```


`GET /explorer/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 33bba9e4-2fb7-4f60-b9ed-8d11e778853e).

Fields: `query`, `chainId`, `order`, `timeline`, `links`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2az POST /feedback

Docs feedback. No key needed.

curl

```
curl -s -X POST "$BASE/feedback" -H "content-type: application/json" \
  -d '{ "page": "first-order", "helpful": true }'
```

TypeScript

```
const res = await fetch(`${BASE}/feedback`, { method: 'POST', headers: { 'content-type':
'application/json' }, body: JSON.stringify({ page: 'first-order', helpful: true }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/feedback", json={"page": "first-order", "helpful": True}).json()
```

Response


```
{
  "received": true
}
```


`POST /feedback` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` 779ad428-18af-4ce0-a4a0-0c9338520f4c).

Fields: `received`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2ba POST /security-reports

Report a security issue. No key needed.

curl

```
curl -s -X POST "$BASE/security-reports" -H "content-type: application/json" \
  -d '{ "summary": "What you found, in a line" }'
```

TypeScript

```
const res = await fetch(`${BASE}/security-reports`, { method: 'POST', headers: { 'content-type': 'application/json' }, body: JSON.stringify({ summary: 'What you found, in a line' }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/security-reports", json={"summary": "What you found, in a line"}).json()
```

Response

```
{
  "received": true,
  "reference": "3932c3dc-c3a4-4ac1-b160-a56a41b7155e",
  "note": "Thank you. A person reads every report."
}
```

`POST /security-reports` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` 1b940435-cb7e-4869-a9f2-253d3e484c09).

Fields: `received`, `reference`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2bb GET /system/status

Every component now, open incidents, upcoming maintenance. No key needed.

curl

```
curl -s "$BASE/system/status"
```

TypeScript

```
const res = await fetch(`${BASE}/system/status`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/system/status").json()
```

Response


```
{
  "status": "operational",
  "checkedAt": "2026-11-13T02:09:35.000Z",
  "components": [
    {
      "id": "api",
      "name": "HTTP API",
      "group": "API",
      "status": "operational",
      "latencyMs": 4127,
      "detail": null,
      "checkedAt": "2026-11-13T02:05:05.072746+00:00"
    },
    {
      "id": "quotes",
      "name": "Prices and quotes",
      "group": "API",
      "status": "operational",
      "latencyMs": 2738,
      "detail": null,
      "checkedAt": "2026-11-13T02:05:05.072746+00:00"
    },
    "…"
  ],
  "openIncidents": [],
  "upcomingMaintenance": []
}
```

`GET /system/status`: a real sandbox answer, 13 November 2026 (`x-request-id` 29fd4367-7957-4a2d-bfef-7c342df4c414), with long fields folded.

Fields: `status`, `checkedAt`, `components`, `openIncidents`, `upcomingMaintenance`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2bc GET /system/history

Each component per day. No key needed.

Parameter	Required	Meaning
`days`	no	

curl

```
curl -s "$BASE/system/history?days=90"
```

TypeScript

```
const res = await fetch(`${BASE}/system/history?${new URLSearchParams({ days: '90' })}`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/system/history", params={"days": "90"}).json()
```

Response


```
{
  "days": 3,
  "components": [
    {
      "id": "api",
      "name": "HTTP API",
      "group": "API"
    },
    {
      "id": "quotes",
      "name": "Prices and quotes",
      "group": "API"
    },
    {
      "id": "solver",
      "name": "Solver",
      "group": "Settlement"
    },
    {
      "id": "bitcoin",
      "name": "Bitcoin network reads and broadcast",
      "group": "Settlement"
    },
    {
      "id": "settler:8453",
      "name": "Base settler",
      "group": "Chains"
    },
    {
      "id": "settler:56",
      "name": "BNB Chain settler",
      "group": "Chains"
    },
    {
      "id": "settler:42161",
      "name": "Arbitrum settler",
      "group": "Chains"
    },
    {
      "id": "settler:43114",
      "name": "Avalanche settler",
      "group": "Chains"
    },
    {
      "id": "settler:4663",
      "name": "Robinhood settler",
      "group": "Chains"
```

```
    },
    {
      "id": "sandbox",
      "name": "Sandbox",
      "group": "Developer"
    }
  ],
  "history": [
    {
      "component": "api",
      "day": "2026-11-13",
      "checks": 26,
      "passed": 26
    },
    {
      "component": "bitcoin",
      "day": "2026-11-13",
      "checks": 26,
      "passed": 26
    },
    "…"
  ]
}
```


`GET /system/history`: a real sandbox answer, 13 November 2026 (`x-request-id` 60b94fc4-6a03-468e-96e4-071c6c835c4f), with long fields folded.

Fields: `days`, `components`, `history`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2bd GET /system/incidents

Incidents with their notes and postmortems. No key needed.

curl

```
curl -s "$BASE/system/incidents"
```

TypeScript

```
const res = await fetch(`${BASE}/system/incidents`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/system/incidents").json()
```

Response

```
{
  "incidents": []
}
```

`GET /system/incidents`: a real sandbox answer, 13 November 2026 (`x-request-id` c3712c31-1528-4eba-8c7b-0548b9922767), with long fields folded.

Fields: `incidents`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2be GET /system/maintenance

The maintenance calendar. No key needed.

curl

```
curl -s "$BASE/system/maintenance"
```

TypeScript

```
const res = await fetch(`${BASE}/system/maintenance`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/system/maintenance").json()
```

Response

```
{
  "maintenance": []
}
```

`GET /system/maintenance`: a real sandbox answer, 13 November 2026 (`x-request-id` 394c4096-4075-46b2-9c17-1580a51ce80a), with long fields folded.

Fields: `maintenance`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2bf GET /system/verified

The last end-to-end run through every failure branch on the sandbox. No key needed.

curl

```
curl -s "$BASE/system/verified"
```

TypeScript

```
const res = await fetch(`${BASE}/system/verified`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/system/verified").json()
```

Response


```
{
  "verified": true,
  "at": "2026-11-12T03:17:05.742743+00:00",
  "source": "server",
  "passed": 11,
  "total": 11,
  "results": [
    {
      "ms": 1302,
      "ok": true,
      "name": "buy with ETH: quote, order, completed",
      "detail": null,
      "requestId": "599658f2-bbbf-480c-a13d-1726cc6965d3"
    },
    {
      "ms": 539,
      "ok": true,
      "name": "gasless buy with USDC",
      "detail": null,
      "requestId": "9bdcd336-53b7-47b1-bb7f-ed1edf7f7b31"
    },
    "…"
  ]
}
```


`GET /system/verified`: a real sandbox answer, 13 November 2026 (`x-request-id` 118d2d2d-cd1b-40d7-ab3d-26fae31bede1), with long fields folded.

Fields: `verified`, `at`, `source`, `passed`, `total`, `results`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2bg GET /system/feed.xml

Incidents as RSS. No key needed.

curl

```
curl -s "$BASE/system/feed.xml"
```

TypeScript

```
const res = await fetch(`${BASE}/system/feed.xml`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/system/feed.xml").json()
```

Response


```
{
  "verified": true,
  "at": "2026-11-12T03:17:05.742743+00:00",
  "source": "server",
  "passed": 11,
  "total": 11,
  "results": [
    {
      "ms": 1302,
      "ok": true,
      "name": "buy with ETH: quote, order, completed",
      "detail": null,
      "requestId": "599658f2-bbbf-480c-a13d-1726cc6965d3"
    },
    {
      "ms": 539,
      "ok": true,
      "name": "gasless buy with USDC",
      "detail": null,
      "requestId": "9bdcd336-53b7-47b1-bb7f-ed1edf7f7b31"
    },
    {
      "ms": 230,
      "ok": true,
      "name": "expired quote: 409 with a fresh quote",
      "detail": null,
      "requestId": "de0bfad6-8f2a-48b0-9407-c47416366a8a"
    },
    {
      "ms": 119,
      "ok": true,
      "name": "over the order size cap: quote shows, order refused",
      "detail": null,
      "requestId": "6b070867-c0ef-46d5-857f-99c9a1e0a4b6"
    },
    {
      "ms": 92,
      "ok": true,
      "name": "larger than can be filled: price shows, no figure",
      "detail": null,
      "requestId": "294edcdb-e73c-4d77-8be9-7a90e68fe7d6"
    },
    {
      "ms": 457,
      "ok": true,
      "name": "fill timeout, then refund",
      "detail": null,
```

```
      "requestId": "b5715d20-cead-4911-a964-ae442cceed21"
    },
    {
      "ms": 341,
      "ok": true,
      "name": "order fails with a reason code",
      "detail": null,
      "requestId": "fdcfdec4-9fb2-4c74-9909-84c9dc454f57"
    },
    {
      "ms": 1204,
      "ok": true,
      "name": "sell BTC: payment instructions, completed",
      "detail": null,
      "requestId": "18052499-9a5b-4f78-af1e-c7238cfcded0"
    },
    {
      "ms": 96,
      "ok": true,
      "name": "statement for this month",
      "detail": null,
      "requestId": "c552f9ef-58f5-4ed3-8c01-4501e559001c"
    },
    {
      "ms": 77,
      "ok": true,
      "name": "webhook endpoint: signed test event is created",
      "detail": null,
      "requestId": "228fe31b-57f4-47df-8c4d-6b8a5ea6ae3b"
    },
    {
      "ms": 1,
      "ok": true,
      "name": "degraded health can be simulated",
      "detail": null,
      "requestId": "79b338c1-ae9a-414c-879d-6162d391ecce"
    }
  ]
}
```


`GET /system/verified`: a real sandbox answer, 13 November 2026 (`x-request-id` 118d2d2d-cd1b-40d7-ab3d-26fae31bede1).

Fields: `verified`, `at`, `source`, `passed`, `total`, `results`.

Errors: none of its own beyond `rate_limited`. See error codes.

10.2bh POST /system/subscribe

Status notices by webhook or email. No key needed.

curl

```
curl -s -X POST "$BASE/system/subscribe" -H "content-type: application/json" \
  -d '{ "kind": "webhook", "target": "https://your.app/maxy/status" }'
```

TypeScript

```
const res = await fetch(`${BASE}/system/subscribe`, { method: 'POST', headers: { 'content-type':
'application/json' }, body: JSON.stringify({ kind: 'webhook', target:
'https://your.app/maxy/status' }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/system/subscribe", json={"kind": "webhook", "target":
"https://your.app/maxy/status"}).json()
```

Response

```
{
  "subscribed": true,
  "kind": "webhook",
  "target": "https://postman-echo.com/post?capture=status",
  "secret": "mxs_… (shown once: store it)",
  "unsubscribeToken": "…",
  "note": "Verify events with Maxy-Signature and this secret, as order webhooks. Shown once."
}
```

`POST /system/subscribe` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` fe352819-d48e-443d-920e-3508c09660a3).

Fields: `subscribed`, `kind`, `target`, `secret`, `unsubscribeToken`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2bi GET /system/unsubscribe

Stop status notices. No key needed.

Parameter	Required	Meaning
`token`	yes	

curl

```
curl -s "$BASE/system/unsubscribe?token=$TOKEN"
```

TypeScript

```
const res = await fetch(`${BASE}/system/unsubscribe?${new URLSearchParams({ token: undefined
})}`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/system/unsubscribe", params={"token": undefined}).json()
```

Response

```
{
  "subscribed": true,
  "kind": "webhook",
  "target": "https://postman-echo.com/post?capture=status",
  "secret": "mxs_… (shown once: store it)",
  "unsubscribeToken": "…",
  "note": "Verify events with Maxy-Signature and this secret, as order webhooks. Shown once."
}
```

`POST /system/subscribe` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` fe352819-d48e-443d-920e-3508c09660a3).

Fields: `unsubscribed`, `subscribed`, `kind`, `target`, `secret`, `unsubscribeToken`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2bj GET /system/confirm

Confirm an email subscription. No key needed.

Parameter	Required	Meaning
`token`	yes	

curl

```
curl -s "$BASE/system/confirm?token=$TOKEN"
```

TypeScript

```
const res = await fetch(`${BASE}/system/confirm?${new URLSearchParams({ token: undefined })}`);
const answer = await res.json();
```

Python

```
answer = requests.get(f"{BASE}/system/confirm", params={"token": undefined}).json()
```

Response

```
{
  "subscribed": true,
  "kind": "webhook",
  "target": "https://postman-echo.com/post?capture=status",
  "secret": "mxs_… (shown once: store it)",
  "unsubscribeToken": "…",
  "note": "Verify events with Maxy-Signature and this secret, as order webhooks. Shown once."
}
```

`POST /system/subscribe` (HTTP 201): a real sandbox answer, 13 November 2026 (`x-request-id` fe352819-d48e-443d-920e-3508c09660a3).

Fields: `confirmed`, `subscribed`, `kind`, `target`, `secret`, `unsubscribeToken`, `note`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

10.2bk POST /sandbox/orders/{id}/advance

Test keys only: move a sandbox order to its next state now, or to the state you name. Needs your key.

curl

```
curl -s -X POST -H "x-maxy-key: $MAXY_API_KEY" "$BASE/sandbox/orders/$ORDER_ID/advance" -H
"content-type: application/json" \
  -d '{ "to": "completed" }'
```

TypeScript

```
const res = await fetch(`${BASE}/sandbox/orders/${orderId}/advance`, { method: 'POST', headers:
{ 'x-maxy-key': KEY, 'content-type': 'application/json' }, body: JSON.stringify({ to:
'completed' }) });
const answer = await res.json();
```

Python

```
answer = requests.post(f"{BASE}/sandbox/orders/{order_id}/advance", headers={"x-maxy-key": KEY},
json={"to": "completed"}).json()
```

Response


```
{
  "orderId": "0x53414e44424f5823936eadc65c68429f1fc465d671b4382365c5ff83ffbf13ae",
  "route": "evm-btc-escrow",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7",
  "txs": [
    {
      "label": "btc_payout",
      "hash": "53414e44424f584ae8dec8623d0c26e0d71d5d285965afaaa487298533b1cdf7"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "advanced": true
}
```


`POST /sandbox/orders/{id}/advance`: a real sandbox answer, 13 November 2026 (`x-request-id` 172f8ed3-4ac3-4e7f-a061-e290c75044a6).

Fields: `orderId`, `route`, `state`, `rawStatus`, `isComplete`, `btcTxId`, `txs`, `refundAt`, `refundAtIso`, `refundAfterBlock`, `externalId`, `externalCustomerId`, `failureReason`, `scenario`, `advanced`.

Errors: 400: `invalid_input`; 401: `auth_required`, `invalid_api_key`; 403: `key_revoked`; 409: `quote_expired`, `quote_mismatch`, `conflict`; 422: `amount_out_of_range`, `order_cap_exceeded`, `insufficient_funds`, `insufficient_liquidity`, `tx_reverted`; 429: `rate_limited`. See error codes.

Next: 10.3 SDK reference.

Last verified: 13 November 2026, against the sandbox.

10.3 SDK reference

After this page you know what `@maxy-labs/sdk` exports, where each part is documented, and which HTTP call sits behind it.

Before you start: nothing. **Time:** 3 minutes.

The reference is written by hand from the source and lives in the SDK chapter. The package is `@maxy-labs/sdk` (MIT licence), installed from maxy.co.

10.3a Packages

Import	What it exports	Reference
`@maxy-labs/sdk/core`	The client over the HTTP API alone, flow helpers, `units`, `createSwapMachine`, the error classes, generated types	the SDK client
`@maxy-labs/sdk/evm`, `/solana`, `/bitcoin`, `/server`, `/testing`	Flow helpers per chain, server helpers, test tools	the SDK client
`@maxy-labs/sdk`	`createMaxyClient`, every class, the signers, `verifyWebhook`, `signWebhook`	core
`@maxy-labs/sdk/swap`	`swap.price`, `quote`, `execute`, `createOrder`, `getStatus`, `waitForCompletion`, `planRefund`, `refund`, `pairs`	swap
`@maxy-labs/sdk/orders`	`orders.list` and the order records	orders
`@maxy-labs/sdk/btc`	Address decoding, Bitcoin payment helpers	btc
`@maxy-labs/sdk/borrow`	MaxyLock loans	borrow
`@maxy-labs/sdk/mint`	Maxy Mint	mint
`maxy` (command)	Every swap, order and key command, with `--json`	the maxy command

---

10.3b Calls and their endpoints

SDK	HTTP
`swap.pairs()`	`GET /pairs`
`swap.price(req)`	`GET /price`
`swap.quote(req)`	`GET /quote`

SDK	HTTP
`swap.createOrder(quote, { evm, btcAddress })`	`POST /orders`, `POST /orders/submit`
`swap.execute(req)`	All of the above, then status until done
`swap.getStatus(order)`	`GET /status`
`swap.waitForCompletion(order)`	`GET /status`, every 4 seconds for up to 30 minutes
`swap.planRefund(...)`, `swap.refund(...)`	`GET /orders/{id}/refund`
`verifyWebhook(body, header, secret)`	Nothing: local
`prices.get`, `quotes.get`, `quotes.batch`, `orders.*`, `status.*`, `addresses.validate`, `limits.get`, `health.get`	One route each: the SDK client

The thin client, the flow helpers, React and Python are on the SDK client.

10.3c Failure shape

Every deliberate failure is a `MaxyError` subclass with `code`, `message`, `status`, `requestId`, `serverCode`, `retryable`, `details`, `docsUrl` and `cause`. Branch on the class or on `code`; `isMaxyError(e, code)` is the guard. The classes are on the SDK client, the codes on error codes.

TypeScript

```
import { createMaxyClient, isMaxyError } from '@maxy-labs/sdk';

const maxy = createMaxyClient(); // reads MAXY_API_KEY; a test key runs the sandbox
try {
  const price = await maxy.swap.price({ from: 'USDC on base', to: 'BTC', amount: '10' });
  console.log(price.formatted?.receive);
} catch (error) {
  if (isMaxyError(error, 'rate_limited')) console.log('wait');
  else throw error;
}
```

10.3d Python and curl

There is no Python package. Python calls the HTTP API with `requests`, and every guide shows it. The same goes for any other language: see the generated API reference.

10.3e Errors on this page

Every SDK failure is a `MaxyError`; its `code` is on error codes.

Next: 10.4 The SDK client.

Last verified: 5 November 2026, against the SDK source and the API it runs.

10.4 The SDK client

After this page you can call every route with one typed method, run a whole swap with one flow helper, and handle every failure by its class.

Before you start: a test key (quickstart). **Time:** 15 minutes.

The package is `@maxy-labs/sdk`, MIT licence, installed from maxy.co: `npm install https://maxy.co/sdk/maxy-labs-sdk-latest.tgz viem` (SHA-256 in https://maxy.co/sdk/SHA256SUMS; to upgrade, install the same URL again).

10.4a What to import

Import	Gives	Depends on	Runs in
`@maxy-labs/sdk/core`	The client below, flow helpers, `units`, `createSwapMachine`, the error classes, generated types	`fetch` only, 17 kB gzipped	Browser, Node 18+, edge
`@maxy-labs/sdk/evm`	`buyBtcWithEvm` taking a viem `WalletClient` or an EIP-1193 provider, `evmWallet`, settler ABI and addresses	`viem` (peer)	Browser, Node
`@maxy-labs/sdk/solana`	`buyBtcWithSolana`, the program addresses	Nothing: the API builds the transactions	Browser, Node
`@maxy-labs/sdk/bitcoin`	`sellBtc`, `bip21Uri`, `validateAddress`, `units`	The Bitcoin libraries	Browser, Node
`@maxy-labs/sdk/server`	`verifyWebhook`, `constructEvent`, `handleOnce`, `exportOrders`	WebCrypto	Node, edge. A browser build refuses this path
`@maxy-labs/sdk/testing`	`createMockFetch`, `errorFixture`, `ERROR_FIXTURES`, `SandboxSimulator`, `mswHandlers`	Nothing	Your tests
`@maxy-labs/sdk`	All of the above, plus `swap`, `borrow`, `mint`, `auth`	The peers in the package	Browser, Node
`@maxy/react`	Hooks: React	`react`, `@maxy-labs/sdk/core`	Browser, server rendering
`maxy` (Python)	The same client in Python: Python	`httpx`	Python 3.10+

10.4b The client

Every method is one route. Anything it does can be done with curl.

TypeScript

```
import { createMaxyClient } from '@maxy-labs/sdk/core';

const maxy = createMaxyClient({ apiKey: process.env.MAXY_API_KEY });

const pairs = await maxy.pairs.list();                 // GET /pairs, kept for 60 seconds
const price = await maxy.prices.get({ from: 'USDC on base', to: 'BTC', amount: '100' });
const quote = await maxy.quotes.get({ from: 'USDC on base', to: 'BTC', amount: '100',
fromAddress: wallet, toAddress: 'bc1q…' });

const signature = await walletClient.signTypedData(quote.signing.payloads[0].typedData);
const order = await maxy.orders.create({ quote, signature, externalId: 'inv-1042' });

for await (const state of maxy.orders.watch(order.orderId)) {
  console.log(state.state, state.rawStatus, state.btcTxId);
}
```

Resource	Methods	Route
`prices`	`get`, `stream(params, onTick)`	`GET /price`, `GET /prices/stream`
`pairs`	`list`, `get(from, to)`, `refresh`	`GET /pairs`
`quotes`	`get`, `batch`, `msUntilExpiry`, `refresh`	`GET /quote`, `POST /quotes/batch`
`orders`	`create`, `submitSignature`, `submit`, `get`, `list`, `iterate`, `exportCsv`, `cancel`, `events`, `refund`, `deposit`, `claim`, `bridge`, `steps`, `follow`, `watch`, `resume`	`/orders…`
`status`	`byTxHash`, `byIntentHash`, `byBitcoinTxid`	`GET /status`
`addresses`	`validate(chain, address)`	`GET /addresses/validate`
`limits`, `health`, `key`	`get`	`GET /limits`, `/health`, `/key`
`webhooks`	`create`, `list`, `remove`, `test`, `events`, `replay`	`/webhooks…`
`sandboxControls`	`advance`, `scenarios`, `rates`	`/sandbox/…`

The full `createMaxyClient` from `@maxy-labs/sdk` has all of these too, beside `swap`, `borrow` and `mint`. On it, `orders.list({ wallet })` with a wallet alone is every product of that wallet; any other

filter reads `GET /orders`.

What every method shares:

- **Last argument.** `{ signal, idempotencyKey, requestId, timeoutMs }`. Your `requestId` becomes the answer's `x-request-id`.
- **What comes back.** The answer's JSON, with `.meta`: `requestId`, `status`, `latencyMs`, `retries`, `rateLimit`. `.meta` stays out of `JSON.stringify`.
- **Lists.** `orders.list` returns `{ data, nextCursor, meta }`. `orders.iterate` walks every page.
- **Amounts.** Decimal strings in the API. `units.toSats`, `units.fromSats`, `units.toSmallest`, `units.fromSmallest` and `units.format` do the arithmetic with no float.
- **Time.** `quotes.msUntilExpiry(quote)` reads the `Date` of Maxy's answers, so a laptop five minutes off still counts down right. `quotes.refresh(quote)` asks the same question again.
- **Config.** `apiKey`, `apiUrl`, `baseUrl`, `fetch`, `timeoutMs` (30 000), `maxRetries` (2), `onRequest`, `onResponse`, `onRetry`, `logger`.

Reliability. Reads, and calls with an idempotency key, retry twice on 429, 502, 503, 504 or a dropped connection, with backoff and jitter, honouring `Retry-After`. `orders.create` sets its key from the order id, so it retries safely; any other POST is sent once unless you give it a key. A burst waits for the minute to turn over instead of collecting 429s. Identical reads in flight share one request. `signal` stops the caller at once and sends nothing further.

Logging. The SDK logs nothing without a `logger`. Keys, signatures and typed data are redacted from what it logs.

A secret key in a browser page throws at construction, with a message saying why. Use a publishable key there (keys). A live key is accepted anywhere; with a `logger`, it warns once in a test run.

10.4c Errors

Every failure is a `MaxyError` subclass with `code`, `status`, `requestId`, `retryable`, `details`, `docsUrl` and `cause`. `isMaxyError(e)` is the one guard you need.

Class	When	Also carries
`AuthError`	`auth_required`, `invalid_api_key`, `key_revoked`, `scope_denied`, `origin_not_allowed`, `sandbox_key_refused`	`check`
`RateLimitError`	429	`retryAfterMs`, `limit`, `remaining`

Class	When	Also carries
`ValidationError`	`invalid_input` and the other 400s, `not_found`	
`AmountOutOfRangeError`	Too small, or over the key's order size cap	`limits { minSats, capUsd }`
`UnfillableError`	`insufficient_liquidity`, `insufficient_funds`, `route_disabled`	No figure, by design
`QuoteExpiredError`	`quote_expired`, `quote_mismatch`	
`OrderStateError`	`order_state`, `conflict`, `not_refundable`	`state`, `allowedStates`
`UserRejectedError`	The wallet declined	The wallet's message
`ChainError`	`tx_reverted`, `wrong_chain`, `signer_unsupported`	`chain`, `txHash`
`NetworkError`, `TimeoutError`	Transport	`attempts`
`AbortedError`	Your `signal` fired	
`ServerError`	5xx, or an answer the SDK does not know	
`WebhookSignatureError`	A bad or stale signature in `verifyWebhook`	`check`: `format`, `timestamp`, `digest`, `body`

`retryable` is true only for 429, 502, 503, 504 and transport failures. The codes are on error codes; `docsUrl` opens the row.

10.4d Errors on this page

Code	Class	Fix
`scope_denied`	`AuthError`	A publishable key reads and quotes; send the order from your server with the secret key
`order_state`	`OrderStateError`	Read `allowedStates`

Next: 10.5 SDK flow helpers, React and Python.

Last verified: 5 November 2026, against the SDK source and the API it runs.

10.5 SDK flow helpers, React and Python

After this page you can run a whole swap with one helper, resume it, use it from React or Python, and test it with no network.

Before you start: the SDK client. **Time:** 10 minutes.

10.5a Flow helpers

Each runs a whole order: quote, checks, signatures, order, escrow if the route has one, then watching to a final state.

TypeScript

```
import { buyBtcWithEvm } from '@maxy-labs/sdk/evm';
import { createMaxyClient } from '@maxy-labs/sdk/core';

const client = createMaxyClient({ apiKey: process.env.MAXY_API_KEY });
const done = await buyBtcWithEvm({
  client,
  wallet: walletClient,                  // a viem WalletClient, an EIP-1193 provider, or your
own signer
  from: 'USDC on base',
  amount: '100',                         // or receive: '0.001' for exact BTC
  recipient: 'bc1q…',
  externalId: 'inv-1042',
  signal,
  onProgress: (state, ctx) => {
    if (state === 'submitting') saveOrderId(ctx.orderId);  // keep it: resume needs it
    render(state);
  },
});
console.log(done.order.state, done.bitcoinTxid);
```

Helper	From	What changes
`buyBtcWithEvm`	`@maxy-labs/sdk/evm` (`@maxy-labs/sdk/core` takes a signer)	The wallet signs EIP-712, and sends the escrow on a route with one
`buyBtcWithSolana`	`@maxy-labs/sdk/solana`	The wallet signs a readable request and then the deposit; the API sends what it signed
`sellBtc`	`@maxy-labs/sdk/bitcoin`	Returns early with `paymentInstructions { address, sats, expiresAt, bip21 }`; `done` follows the rest

Progress. The same words on each: `quoting`, `checking`, `awaiting_approval`, `awaiting_signature`, `submitting`, `awaiting_payment`, `awaiting_fill`, `broadcast`, `confirming`, then `completed`, `refundable`, `refunded`, `cancelled`, `expired` or `failed`. A helper stops at `refundable`: nothing more happens without the user.

Resume. A helper stores nothing. Keep the order id when `submitting` fires and pass `resume: { orderId }`, in this process or another, to pick up from there. `orders.resume(orderId)` is the same watcher without a helper.

A page with a publishable key cannot place orders, so pass your server's route: `placeOrder: (input) => fetch('/api/maxy/orders', …)` and `submitOrder`. 4.4 DeFi or DEX frontend shows the whole route.

Your own signing system. `createSwapMachine()` is the same states and allowed moves with no wallet and no network, for a custodial platform that drives each step itself. A move its table does not allow throws `OrderStateError`.

10.5b React

`@maxy/react` is hooks over the core: no styles, no wallet, safe during server rendering, and it works beside wagmi and the Solana wallet adapter without wrapping either.

```
import { MaxyProvider, useQuote, useBuyBtc } from '@maxy/react';
import { evmWallet } from '@maxy-labs/sdk/evm';

<MaxyProvider config={{ apiKey: 'mxk_pk_test_…' }}><Buy /></MaxyProvider>;

function Buy() {
  const { data: walletClient } = useWalletClient();                  // wagmi
  const { quote, secondsLeft } = useQuote({ from: 'USDC on base', to: 'BTC', amount: '100',
fromAddress: walletClient?.account.address, toAddress: 'bc1q…' });
  const buy = useBuyBtc();
  return <button onClick={() => buy.start({ wallet: evmWallet(walletClient), from: 'USDC on
base', amount: '100', recipient: 'bc1q…', placeOrder })}>
    {buy.state === 'idle' ? `Buy ${quote?.formatted?.receive} (${secondsLeft} s)` : buy.state}
  </button>;
}
```

Hook	Gives
`usePairs()`	`data`, `error`, `loading`; kept 60 s
`usePrice(params)`	`data`, refreshed every 5 s while mounted
`useQuote(params, { autoRenew })`	`quote`, `secondsLeft` from Maxy's clock; renewed 5 s before it closes

Hook	Gives
`useBuyBtc()`, `useBuyBtcWithSolana()`	`state`, `progress`, `error`, `result`, `orderId`, `start`, `retry`, `abort`, `resume`
`useSellBtc()`	The same, and `paymentInstructions` as soon as the payout is locked
`useOrder(id)`	The order, live, until it ends

10.5c Python

The `maxy` package has the same resources and method names, in snake_case, with `Maxy` and `AsyncMaxy`. It installs from maxy.co (`pip install https://maxy.co/sdk/maxy-latest.tar.gz`, MIT licence) and needs `httpx`. Flow helpers that need a browser wallet stay in TypeScript. Answers are dicts with `.meta`.

Python

```
from maxy import Maxy, webhooks

maxy = Maxy()  # reads MAXY_API_KEY
quote = maxy.quotes.get("ETH on base", "BTC", "0.01", from_address=wallet, to_address="bc1q…")
signature = your_signer.sign_typed_data(quote["signing"]["payloads"][0]["typedData"])
order = maxy.orders.create(quote["order"], signature=signature, external_id="inv-1042")

for state in maxy.orders.watch(order["orderId"]):
    print(state["state"], state.get("btcTxId"))

event = webhooks.verify(raw_body, signature_header, secret)  # the same checks as TypeScript,
byte for byte
```

10.5d Testing your integration

`@maxy-labs/sdk/testing` runs your code with no network: the same envelopes and headers as the API, one fixture per error code, and a simulator that walks an order through any states.

TypeScript

```
import { createMaxyClient } from '@maxy-labs/sdk/core';
import { SandboxSimulator, createMockFetch, errorFixture } from '@maxy-labs/sdk/testing';

const sim = new SandboxSimulator();
const id = sim.order(['pending', 'in_progress', 'refundable', 'refunded']);
const client = createMaxyClient({ fetch: createMockFetch({ simulator: sim }).fetch });

sim.next(id, 'refundable');
await client.orders.get(id);                                    // state: refundable

const down = createMaxyClient({ fetch: createMockFetch({ failWith: 'rate_limited' }).fetch,
maxRetries: 0 });
```

With Mock Service Worker, `mswHandlers({ http, HttpResponse }, { simulator: sim })` gives handlers for the same answers.

10.5e Errors on this page

Code	Class	Fix
`scope_denied`	`AuthError`	A publishable key cannot place orders: pass `placeOrder` to your server's route
`order_state`	`OrderStateError`	A move the machine's table does not allow: read `allowedStates`

Next: 10.6 Error codes.

Last verified: 5 November 2026, against the SDK source and the API it runs.

10.6 Error codes

This page is for integrators writing error handling: every deliberate failure is a `MaxyError` with a stable `code`, and this page says what each code means, whether to retry, and what was spent.

TypeScript

```
// fragment: `maxy`, `quote`, `evm` and `btcAddress` come from the swap guide
import { isMaxyError } from '@maxy-labs/sdk';

try {
  await maxy.swap.createOrder(quote, { evm, btcAddress });
} catch (error) {
  if (!isMaxyError(error)) throw error;
  switch (error.code) {
    case 'user_rejected': /* the user said no */ break;
    case 'quote_expired': /* get a new quote */ break;
    case 'order_cap_exceeded': console.log(error.message); break; // 'Order size is capped for
this API key. Try $100 or less.'
    default: console.error(error.code, error.serverCode, error.message);
  }
}
```

Fields: `code` (below; stable), `message` (human text; may change), `status` (HTTP status when it came from a server), `serverCode` (the server's own code, kept verbatim; not covered by versioning), `retryable`, `details` (free-form per code; not covered by versioning), `cause`. Read the documented `details` fields with the typed readers `retryAfterSecondsOf`, `unlocksAtOf` and `preparedPostOf` rather than casting.

10.6a HTTP API error codes

Every error from the HTTP API is `{ "error": { "code", "message", "details"?, "retryable", "docs" }, "requestId" }`, with the same id in the `x-request-id` header. `docs` links the code's own row on this page. The SDK raises a `MaxyError` subclass (the classes) with the same `code` in `error.code`, and the same link in `docsUrl`; a code the HTTP API returns is also in the SDK's table below.

Code	HTTP	Retryable	Raised by	Cause	What to do
`invalid_input`	400	no	any route	A missing or malformed parameter or body: no addresses on a quote, a malformed `txHash` or `orderId`, an order not sent back unchanged	Fix the request; the message names the field
`unsupported_chain`	400	no	price, quote, status	A chain Maxy does not swap on	Pick from `GET /chains`
`unsupported_route`	400	no	price, quote, orders	A pair Maxy does not swap	Pick from `GET /pairs`
`wallet_mismatch`	400	no	`POST /orders`, `POST /orders/{id}/steps`	A signature that is not the wallet's the order names	Sign with the wallet the quote names
`verification_failed`	400	no	`POST /orders`, `POST /orders/{id}/steps`	The order's Bitcoin script or address does not match its intent, or a Solana transaction is not the deposit Maxy built	Send the order unchanged; change nothing but the signature
`auth_required`	401	no	every route but `/`, `/health`, `/openapi.json`, the status page and `POST /sandbox/keys`	No key	Send `x-maxy-key` (a test key: `POST /sandbox/keys`)
`invalid_api_key`	401	no	every route	A key Maxy did not issue, or a malformed one	Check the key: `mxk_live_` and 32 letters and digits
`invalid_ticket`	401	no	`GET /orders/{id}/events/stream`, `GET /prices/stream` with `?ticket=`	The stream ticket expired (they last 10 minutes), was changed, or is for another order or pair	Ask a new one: `POST /orders/{id}/stream-ticket` or `POST /prices/stream-ticket`
`key_revoked`	403	no	every route	The key was revoked (within about a minute)	Use your new key
`scope_denied`	403	no	orders, webhooks, your order list	A publishable key asked for something only a secret key can do	Send it from your server with the secret key

Code	HTTP	Retryable	Raised by	Cause	What to do
`origin_not_allowed`	403	no	any route, with a publishable key	The page's origin is not on the key's list, or the call has no origin (a server)	Ask Maxy to add the origin; from a server, use the secret key
`sandbox_key_refused`	403	no	functions other than the HTTP API	A test key (`mxk_test_…`) was sent to another Maxy function (a function on an older build answers `401 invalid_api_key`). Nothing was done	Test keys work only on `/functions/v1/maxy-api`; the SDK does this for you
`checkout_signature_invalid`	403	no	`GET /checkout/verify`	The hosted checkout URL was changed after signing, its key was revoked, or its publishable key belongs to someone else	Sign a new URL with `signCheckoutUrl` on your server
`checkout_expired`	403	no	`GET /checkout/verify`	The URL is past its `exp`	Sign a new URL; links last an hour unless you ask for longer
`not_found`	404	no	unknown routes, `status?txHash=`, webhooks	No such route (the message suggests the right one), no order with that transaction, no webhook with that id	Check the path or id; a brand-new transaction may need a few seconds
`quote_expired`	409	no	`POST /orders`, `POST /orders/submit`, deposit, steps	The quote is older than 60 seconds, or the intent's deadline (2 hours) passed; selling, the commit deadline or the deposit window passed; on Solana, the signed request or the deposit's blockhash expired	Get a new quote and sign again. Selling: do not pay

Code	HTTP	Retryable	Raised by	Cause	What to do
`quote_mismatch`	409	no	`POST /orders`	The escrow would lock more than the quote and its price band allow, or a Solana offer is more than 0.1% below the quote	Get a new quote
`conflict`	409	no	`POST /webhooks`, `POST /orders`, deposit, steps	A sixth webhook endpoint (5 per key); the solver could not lock the payout; the sell is already paid out; the Solana offer is no longer open, or not in the state this stage needs (`broadcast` before `PaymentLocked`)	Delete an endpoint; otherwise get a new quote, and do not pay
`idempotency_conflict`	409	no	`POST /orders`, `/orders/submit`, `/orders/{id}/signature`, `/orders/{id}/steps`	This `Idempotency-Key` was used within a day for a different request	Use a new key for a new order
`order_state`	409	no	`POST /orders/{id}/cancel`, `POST /sandbox/orders/{id}/advance`	Not allowed in the order's state	Read `details.state` and `details.allowedStates`; a committed order refunds at `GET /orders/{id}/refund`
`amount_out_of_range`	422	no	price, quote, orders	Below the smallest order (546 sats after costs)	Raise the amount
`order_cap_exceeded`	422	no	`POST /orders`	Above your key's order size cap; quotes flag it in `issues.orderCap`. There is no other per-order maximum	Order less, or ask Maxy to raise the cap
`insufficient_funds`	422	no	`POST /orders` (quotes report `issues.balance`)	The wallet holds less than the order needs	Fund the wallet

Code	HTTP	Retryable	Raised by	Cause	What to do
`insufficient_liquidity`	422	no	`POST /orders` (prices show the amount; a quote for an order Maxy cannot fill now answers 200 with `liquidityAvailable: false` and no steps)	Maxy cannot fill this order now. No figure is given	A smaller amount, or later
`tx_reverted`	422	no	`POST /orders`, `POST /orders/{id}/steps`	The transaction would revert (a Solana deposit is simulated before it is sent)	Read the message
`deposit_address_unverified`	422	no	`GET /orders/{id}/deposit`	The escrow on chain does not match this order	**Do not pay.** Get a new quote
`server_rejected`	422	no	`POST /orders`	The solver or market maker refused the order; the message says why	Read the message. Selling: do not pay
`rate_limited`	429	yes	every route	Over 600 calls a minute per key (120 per IP on the routes that take no key); selling, the wallet already has three unpaid sell orders open or the solver is busy	Wait `Retry-After` seconds
`internal_error`	500	yes	any route	Something failed on Maxy's side	Retry; if it repeats, send Maxy the `requestId`
`not_implemented`	501	no	webhooks, order list	The server runs without that feature	Tell Maxy
`service_unavailable`	503	yes	any route with a test key	The key could not be checked just now. Nothing was done	Retry
`timeout`	504	yes	any route	A service behind the API did not answer in time	Retry; read the order first if it placed one

Code	HTTP	Retryable	Raised by	Cause	What to do
`bad_response`, `http_error`, `network_error`	502	see `retryable`	any route that reaches the solver or a chain	A service Maxy depends on failed or answered something unusable. Nothing was signed or locked for you	Retry when `retryable` is `true`; if it repeats, send Maxy the `requestId`
`route_disabled`	503	sells and Solana: yes	quote, orders	The route is off right now (a paused settler, a token not allowed)	Another route, or later

10.6b Codes

"What was spent" says what may already have happened when the error is thrown: **nothing**; **gas** (a transaction was sent and may have cost gas); or **in flight** (funds may have moved; check on chain before acting).

Code	Meaning	Retryable	What was spent	What to do
`network_error`	A host could not be reached	yes	nothing	Retry with backoff
`timeout`	No answer in time, or a transaction not mined within the wait	yes	in flight, for a sent transaction	Retry a read; for a sent transaction, look it up before resending
`http_error`	HTTP 5xx or another unexpected status	5xx: yes	nothing	Retry later
`bad_response`	A server answered something the SDK cannot use (missing fields, implausible price)	once	nothing	Retry once with fresh data (a new quote); report it if it repeats
`rate_limited`	HTTP 429	yes	nothing	Wait `retryAfterSecondsOf(error)` seconds when present, else back off (limits)
`not_found`	HTTP 404	no	nothing	Check the id
`conflict`	The state changed, or an idempotency key was reused with a different body	no	nothing	Reload the order before acting

Code	Meaning	Retryable	What was spent	What to do
`invalid_input`	A parameter is missing or malformed	no	nothing	Fix the call
`config_missing`	A required configuration value is absent	no	nothing	Configure it
`missing_dependency`	A Solana order without `@solana/web3.js` installed (or passed as `solanaWeb3` in a browser)	no	nothing	`npm install @solana/web3.js`
`auth_required`	No session, or the server rejected it (401, 403)	no	nothing	Sign in again (getting started)
`wallet_mismatch`	A signature came from another account, or the session belongs to another wallet	no	nothing	Use the same wallet throughout
`wrong_chain`	The wallet is on another chain and could not be switched	no	nothing	Switch the wallet
`unsupported_chain`	The chain has no deployment for this product	no	nothing	
`unsupported_route`	No route for this asset pair or token	no	nothing	Use another route (for example `cross-evm` first)
`route_disabled`	The route exists but is off right now (settler paused, token not allowlisted, vault paused, keeper not healthy, HTLC mint mode not deployed)	later	nothing	Try later or another route (operations)
`user_rejected`	The user rejected a wallet prompt	no	nothing	
`signer_unsupported`	The signer cannot do what is needed (no message signing, wrong address type, unusable signature)	no	nothing	Use another wallet (signers)

Code	Meaning	Retryable	What was spent	What to do
`tx_reverted`	A simulation or a mined transaction reverted	no	nothing for a simulation; gas for a mined revert	Read `message`
`verification_failed`	The SDK refused data from a server or wallet that did not match its own computation	no	nothing	Do not proceed; retry with fresh data; report it if it repeats
`deposit_address_unverified`	A BTC to EVM payout is not escrowed in the settler for you, or a mint deposit is not registered	no	nothing	**Do not pay it**
`quote_expired`	The quote, deposit window or terms expired	no	nothing	Get a new quote
`quote_mismatch`	The quote does not match the vault, market or terms asked for	no	nothing	Get a new quote
`insufficient_liquidity`	Unfillable right now: the order is larger than Maxy can fill at the moment (the message names no limit)	later	nothing	Try a smaller amount
`insufficient_funds`	The wallet's balance or Bitcoin coins do not cover the order	no	nothing	Fund the wallet
`amount_out_of_range`	Outside the product's limits	no	nothing	Change the amount (limits)
`order_cap_exceeded`	"Order size is capped for this API key. Try $100 or less.": the order is above your key's order size cap; prices and quotes still answer	no	nothing	Order less, or ask Maxy to raise the cap (limits)
`key_revoked` (HTTP 403, `server_rejected`)	The API key was revoked	no	nothing	Ask Maxy for a new key
`not_refundable`	Nothing to refund yet, or at all	later	nothing	Wait until `unlocksAtOf(error)`, or check the order

Code	Meaning	Retryable	What was spent	What to do
`aborted`	Your `AbortSignal` fired	no	nothing further was sent	Start again if you still want it
`sandbox_refused`	A test key asked for something with no sandbox form: a loan, a mint, sign-in or a low-level order builder	no	nothing	Use `swap.quote`, `createOrder`, `execute`, `getStatus` and `refund`, or a live key
`server_rejected`	A server refused the request (4xx), or every Bitcoin node rejected a transaction	no	nothing	Read `message` and `serverCode`

Two outcomes are returned rather than thrown because they are not failures: a Bitcoin broadcast with `outcome.status === 'ambiguous'` (no node confirmed it; the payment may be in flight; never pay again before looking the txid up), and a swap `getStatus` 404, which is reported as `pending`.

10.6c Server codes you may see in `serverCode`

10.6c.1 Maxy API (swaps)

`serverCode` or message	Code	Meaning
`insufficient_liquidity`	`insufficient_liquidity`	A BTC to EVM order larger than the solver can pay
`Rate limit exceeded`	`rate_limited`	Per-IP or per-wallet limit
Validation messages such as `intentSignature must be valid` or `escrowTxHash must be a valid tx hash`	`server_rejected`	A request field failed validation
HTTP 409 with `nextAction`	`conflict`	The solver wants another flow
HTTP 422 with `retryable: false`	`server_rejected`	The commit would revert on chain
`serverCode` `order_cap_exceeded` (HTTP 422)	`server_rejected`	An order sent without the SDK's own check is above your API key's order size cap; the message says the cap ("Try $100 or less.") and `details.body.error.capUsd` holds it (limits)

`serverCode` or message	Code	Meaning
HTTP 503 `liquidity_check_unavailable` or `pricing unavailable`	`http_error`	Do not send BTC; retry later

10.6c.2 Maxy API (borrow)

`serverCode`	Code
A local plan check (`borrower_mismatch`, `terms_hash_mismatch`, `quote_mismatch` from the plan check, `script_reuse`, ...), `presign–missing`, a Bitcoin transaction check code	`verification_failed`
`quote_mismatch` (quote response)	`quote_mismatch`
`quote_expired`, `requote`, `expired`	`quote_expired`
`vault_capacity`	`insufficient_liquidity`
`insufficient_btc`, `insufficient–funds`, `unconfirmed–utxo`, `no_gas`	`insufficient_funds`
`principal_below_min`, `principal_above_max`, `collateral_below_min`	`amount_out_of_range`
`disabled`, `commits_paused`, `vault_paused`, `verifier_unhealthy`, `verifier_floor`, `keeper_unhealthy`, `keeper_not_live`, `lender_key_unavailable`, `btc_network_disabled`, `vault_misconfigured`	`route_disabled`
`wallet_limit`, `wallet_unfunded_limit`, `btc_address_in_use`, `duplicate_terms`, `already_signed`, `nonce_in_use`, `not_cancellable`, return refusals (`not_committed`, `presign_in_use`, `loan_closed`), `RepayTooLate`	`conflict`
Return validation codes, `bad_signature`, `borrower_blocked`, a terminal request status in `waitForCommit`	`server_rejected`
nothing to claim	`not_found`
secret not public yet, recovery time not reached	`not_refundable`

10.6c.3 Maxy Mint API

`serverCode`	Code	Note
`IDEMPOTENCY_CONFLICT`, `STATE_CONFLICT`, `EPOCH_CLOSED`, `ADMISSION_CLOSED`, other 409	`conflict`	Reload the order; never rotate the idempotency key to get past it. Also thrown when the API already reports a payment.
`ORDER_NOT_READY`	`conflict` (retryable)	Wait and retry

`serverCode`	Code	Note
`RATE_LIMITED`, 429	`rate_limited`	
`NOT_FOUND`, 404	`not_found`	
other 4xx	`server_rejected`	
5xx	`http_error`	
deposit not registered, or the address shown is not the signed one	`deposit_address_unverified`	Do not pay
nothing to refund yet	`not_refundable`	retryable later
epoch closed, contract paused, HTLC mode not deployed	`route_disabled`	
refund or redemption quote differs from the request	`quote_mismatch`	

10.6d Where next

- Limits and operations: the causes behind `rate_limited`, `amount_out_of_range` and `route_disabled`.
- Changelog: what versioning covers.
- 10.13 FAQ.

Next: 10.7 Webhook events.

Last verified: 7 November 2026, against the SDK source and the API it runs.

10.7 Webhook events

After this page you have every event's schema, its ordering and retry rules, and a delivery your verifier can test against.

Before you start: nothing. **Time:** 5 minutes.

10.7a Events

Event	When it fires	`data`
`order.updated`	An order's `state` or `btcTxId` changed	The order's status, as `GET /status` returns it, plus `previousState`
`webhook.test`	You called `POST /webhooks/{id}/test`	`webhookId`, `message`

Every state on order states arrives as one `order.updated`: `pending`, `awaiting_deposit`, `in_progress`, `completed`, `refundable`, `refunded`, `cancelled`, `expired`, `failed`.

Register an endpoint and read its log:

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/webhooks/$WEBHOOK_ID/events"
```

TypeScript

```
const { events } = await fetch(`${BASE}/webhooks/${webhookId}/events`, { headers: { 'x-maxy-key': KEY } }).then((r) => r.json());
```

Python

```
events = requests.get(f"{BASE}/webhooks/{webhook_id}/events", headers={"x-maxy-key": KEY}).json()["events"]
```

10.7b Envelope

Field	Type	Meaning
`id`	string	The event id, `evt_` and 24 hex digits. Dedupe on it
`type`	string	`order.updated` or `webhook.test`

Field	Type	Meaning
`createdAt`	string	When the event was made, RFC 3339 in UTC
`data`	object	Below
`sandbox`	boolean	`true` on a sandbox event; absent on a live one

10.7c `order.updated` data

Field	Type	Meaning
`orderId`	string	The order id
`route`	string	`evm-btc-escrow`, `evm-btc-gasless`, `btc-evm`, `sol-btc`, `btc-sol` or `cross-evm`
`chainId`	number or null	The EVM chain the order was paid on
`state`	string	One of the nine states
`previousState`	string or null	The state Maxy reported before this one
`rawStatus`	string	The solver's own status, for support
`isComplete`	boolean	`true` in a final state
`btcTxId`	string or null	The Bitcoin payment, once sent
`txs`	array	`{ label, hash }` for each transaction of the order
`sandbox`, `scenario`		On sandbox events only

10.7d Headers

Header	Value
`Maxy-Signature`	`t=<unix seconds>,v1=<hex>`: `v1` is HMAC-SHA256 of `<t>.<raw body>`, keyed with the endpoint secret
`Maxy-Event-Id`	The event `id`
`Maxy-Event-Type`	The event `type`
`User-Agent`	`Maxy-Webhooks/1`

10.7e Ordering and retries

- Delivery is at least once. Dedupe on the event `id`.
- Up to 8 events go out at once, so two events for one order can arrive out of order. Sort by `createdAt`, or ignore an event whose `previousState` is not the state you hold.
- Answer any `2xx` within 10 seconds. Anything else is retried 8 times (30 s, 2 min, 10 min, 30 min, 1 h, 3 h, 6 h, 12 h).
- Retries stop after 9 attempts, about 23 hours.
- An order is followed until a final state, or for 3 days. Events are kept for 30 days.

10.7f Example deliveries

Each is signed with the SDK's own signer and the published test secret below. It is the same code Maxy's servers sign with. Feed `body` and the header to your verifier at the vector's own time.

`order.updated`

Response


```
{
  "secret": "whsec_maxy_test_vector_not_a_real_secret_000000",
  "timestamp": 1793644197,
  "body": "
{\"id\":\"evt_92c8c63c194b601ce42da1a3\",\"type\":\"order.updated\",\"createdAt\":\"2026-11-
02T18:29:57.000Z\",\"data\":
{\"orderId\":\"0xed3985cf8b6dcc9a30448c9faaefbaf3a2f2ef9c06f50ce1f46dd955212ce2ea\",\"route\":\"evm
btc-
gasless\",\"chainId\":8453,\"state\":\"completed\",\"previousState\":\"in_progress\",\"rawStatus\":
[{\"label\":\"escrow\",\"hash\":\"0xf103819940fef19e88f9c0eafb24a52bad8de86c277e18e8e2e60a8594c7f8d
{\"label\":\"btc_payout\",\"hash\":\"6afc6d649cef90d92d8413ccc12aeb14945c92e5b63db9724cf12b6ef985ea
{\"label\":\"settlement\",\"hash\":\"0xef66f624a34cff33a5d5660e1666e15669bc02dfc8b0a27fa827e8b268e7
  "headers": {
    "content-type": "application/json",
    "user-agent": "Maxy-Webhooks/1",
    "maxy-event-id": "evt_92c8c63c194b601ce42da1a3",
    "maxy-event-type": "order.updated",
    "maxy-signature":
"t=1793644197,v1=3dcfb13254225870c0d80924d3a267d926286c09d26fb054358bb793361aaf68"
  }
}
```


`webhook.test`

Response

```
{
  "secret": "whsec_maxy_test_vector_not_a_real_secret_000000",
  "timestamp": 1793728879,
  "body": "
{\"id\":\"evt_e7bb43c01368ea59dd85712b\",\"type\":\"webhook.test\",\"createdAt\":\"2026-11-
03T18:01:19.000Z\",\"data\":{\"webhookId\":\"0d625e15-f920-4f77-a044-
f4a9ddd040c6\",\"message\":\"A test event from Maxy. Verify its signature with your endpoint
secret.\"}}",
  "headers": {
    "content-type": "application/json",
    "user-agent": "Maxy-Webhooks/1",
    "maxy-event-id": "evt_e7bb43c01368ea59dd85712b",
    "maxy-event-type": "webhook.test",
    "maxy-signature":
"t=1793728879,v1=7aed89894e2107afb1d754dde9d5b6cca24a55777e7a6dbb4f79454670d0a24c"
  }
}
```

`verifyWebhook` accepts each one at its timestamp, refuses it with one byte changed, and refuses it 301 seconds later. Run `verify-webhook.mjs` or `verify_webhook.py` from your kit's `examples` folder, or:

TypeScript

```
import { verifyWebhook } from '@maxy-labs/sdk';
import vector from './webhook-test-vector.json' with { type: 'json' };

const event = await verifyWebhook(vector.body, vector.headers['maxy-signature'], vector.secret,
{ nowSec: vector.timestamp });
console.log(event.type); // order.updated
```

Python

```
from verify_webhook import verify
import json
vector = json.load(open("webhook-test-vector.json"))
assert verify(vector["body"].encode(), vector["headers"]["maxy-signature"], vector["secret"],
now=vector["timestamp"])
```

10.7g A real delivery log

`GET /webhooks/{id}/events` shows the last 25 `order.updated` events of an endpoint. A sandbox order, delivered to a public echo service:

Response


```
{
  "events": [
    {
      "id": "evt_0d4b0ca996c26c37e45f72c6_replay",
      "type": "order.updated",
      "createdAt": "2026-11-13T02:08:30.84+00:00",
      "attempts": 0,
      "deliveredAt": null,
      "failedAt": null,
      "lastStatus": null,
      "lastError": null,
      "data": {
        "txs": [],
        "route": "evm-btc-gasless",
        "state": "in_progress",
        "btcTxId": null,
        "chainId": 8453,
        "orderId": "0x53414e44424f58bd874a9c3b8b3e2b671f7f15d92a42a9c9c34bd5d8079ca56b",
        "sandbox": true,
        "scenario": "webhook_replay",
        "rawStatus": "SANDBOX_FILLING",
        "isComplete": false,
        "previousState": "pending"
      }
    },
    "…"
  ],
  "sandbox": true
}
```


`GET /webhooks/{id}/events`: a real sandbox answer, 13 November 2026 (`x-request-id` bff7e43b-d7ba-4e1b-9560-6f372d7b8f8d), with long fields folded.

10.7h Errors on this page

Code	HTTP	Cause	Fix
`not_found`	404	No webhook with that id for your key	Check the id
`auth_required`	401	Webhook routes need your key	Send `x-maxy-key`

Next: 10.8 Order states.

Last verified: 13 November 2026, against the sandbox.

10.8 Order states

After this page you have one table for every state: its meaning, who acts next, how long it lasts, its webhook and its words.

Before you start: nothing. **Time:** 3 minutes.

The diagram on order lifecycle is drawn from this table. Each name is spelled as the API returns it.

`state`	Who acts next	Typical duration	Final	Webhook	Show the user
`pending`	The solver	Seconds	no	`order.updated`	"Waiting for your transaction to confirm on Base."
`awaiting_deposit`	The user (selling)	Until the deposit window closes: live orders, 19 Bitcoin blocks, about 3 hours	no	`order.updated`	"Send exactly 0.0005 BTC to the address below, in one payment."
`in_progress`	The solver	Buying: seconds to a minute. Selling: one confirmation, then about a minute	no	`order.updated`	"Your BTC is on its way." Selling: "Payment seen. Waiting for 1 confirmation."
`completed`	Nobody		yes	`order.updated`	"0.00025282 BTC is on its way to your wallet." Selling: "Done. 38.01 USDC is in your wallet."
`refundable`	The user	Until the user sends the refund	no: nothing happens without the user	`order.updated`	"The swap did not complete. Your 0.01 ETH is safe in the swap contract. [Get it back]"
`refunded`	Nobody		yes	`order.updated`	"0.01 ETH is back in your wallet."
`cancelled`	Nobody		yes	`order.updated`	"Nothing was sent."
`expired`	Nobody; a late or mismatched sell payment goes to Maxy support		yes	`order.updated`	"The order expired before it started. Nothing was sent."
`failed`	Maxy support		yes	`order.updated`	"Your coins are safe. Contact support with order 0xa46d...0c50."

Polling reads the same names: `GET /status`, `GET /orders/{id}` and every `order.updated` carry one `state`. The solver's own status is `rawStatus`, listed per state below.

10.8a Each state, from the sandbox

curl

```
curl -s "$BASE/orders/$ORDER_ID"
```

TypeScript

```
const order = await fetch(`${BASE}/orders/${orderId}`).then((r) => r.json());
```

Python

```
order = requests.get(f"{BASE}/orders/{order_id}").json()
```

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "state": "pending",
  "rawStatus": "SANDBOX_PENDING",
  "isComplete": false,
  "btcTxId": null,
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success"
}
```


`GET /status`: a real sandbox answer, 13 November 2026 (`x-request-id` 85b0393e-2afe-4485-9f3b-2cbcb02b3240).

Response


```
{
  "orderId": "0x53414e44424f58ce8483f271a52bb64d58d6c8b34209d34c0733d5d2556ef926",
  "route": "btc-evm",
  "state": "awaiting_deposit",
  "rawStatus": "SANDBOX_AWAITING_DEPOSIT",
  "isComplete": false,
  "btcTxId": null,
  "txs": [],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "refundAt": null,
  "refundAtIso": null,
  "refundAfterBlock": null,
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 96078ee5-5b05-4914-a0af-f1ed673794b2).

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "state": "in_progress",
  "rawStatus": "SANDBOX_FILLING",
  "isComplete": false,
  "btcTxId": null,
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success",
  "refundAt": null,
  "refundAtIso": null,
  "refundAfterBlock": null,
  "externalId": null,
  "externalCustomerId": null
}
```

`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 798bcbbe-a611-460e-9ace-d71b44de4264).

Response


```
{
  "orderId": "0x53414e44424f589aa64a8c0ee6e1cc6c96f4549a46f8dbcbaef270a82a2333c3",
  "route": "evm-btc-escrow",
  "state": "completed",
  "rawStatus": "SANDBOX_COMPLETED",
  "isComplete": true,
  "btcTxId": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9",
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f587275400aa546ca2ac2b84bcc59cfd39e276e393980b96bdb13"
    },
    {
      "label": "btc_payout",
      "hash": "53414e44424f58180157a60e949ed223ebb533d0b5e3062f5b8138765c962bb9"
    }
  ],
  "failureReason": null,
  "sandbox": true,
  "scenario": "success"
}
```


`GET /status`: a real sandbox answer, 13 November 2026 (`x-request-id` e828f87c-73d8-4eee-8ff3-ea5539d44411).

Response


```
{
  "orderId": "0x53414e44424f58ec837ff3f2e3b5ff0db94324f8ff30f5e2d799de1f997273cd",
  "route": "evm-btc-escrow",
  "state": "refundable",
  "rawStatus": "SANDBOX_REFUNDABLE",
  "isComplete": false,
  "btcTxId": null,
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f58a6b9ebf67d835aeb77a71569a58bbd2b187dab0cf0ec272259"
    }
  ],
  "failureReason": {
    "code": "no_solver_commit",
    "message": "No solver committed before the escrow deadline. The escrow goes back to the
user."
  },
  "sandbox": true,
  "scenario": "refund_available",
  "refundAt": 1794535573,
  "refundAtIso": "2026-11-13T02:06:13.000Z",
  "refundAfterBlock": null,
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 43a5095c-8188-4b3e-9950-c54f470f71af).

Response


```
{
  "orderId": "0x53414e44424f58ec837ff3f2e3b5ff0db94324f8ff30f5e2d799de1f997273cd",
  "route": "evm-btc-escrow",
  "state": "refunded",
  "rawStatus": "SANDBOX_REFUNDED",
  "isComplete": true,
  "btcTxId": null,
  "txs": [
    {
      "label": "escrow",
      "hash": "0x53414e44424f58a6b9ebf67d835aeb77a71569a58bbd2b187dab0cf0ec272259"
    }
  ],
  "failureReason": {
    "code": "refunded",
    "message": "The escrow went back to the user."
  },
  "sandbox": true,
  "scenario": "refund_available",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 6403ade0-83fc-46f7-b361-43ffa7acc691).

Response


```
{
  "orderId": "0x53414e44424f589cb7e19cf09c6323eece6c9d1b438a6a719e4d720f5d95ebed",
  "route": "evm-btc-gasless",
  "state": "cancelled",
  "rawStatus": "SANDBOX_CANCELLED",
  "isComplete": true,
  "btcTxId": null,
  "txs": [],
  "failureReason": {
    "code": "cancelled",
    "message": "The order was cancelled before a solver committed."
  },
  "sandbox": true,
  "scenario": "order_cancelled",
  "externalId": null,
  "externalCustomerId": null
}
```

`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 4daabb3f-712b-44b3-96f4-188cfdb0a028).

Response


```
{
  "orderId": "0x53414e44424f58c5dd7b697cd10e8d2586ce54ea0b523c05728a4964031aa37d",
  "route": "evm-btc-gasless",
  "state": "expired",
  "rawStatus": "SANDBOX_EXPIRED",
  "isComplete": true,
  "btcTxId": null,
  "txs": [],
  "failureReason": {
    "code": "order_expired",
    "message": "The order lapsed before anything was locked. Nothing moved."
  },
  "sandbox": true,
  "scenario": "order_expired",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` 54ccf710-8e88-46ea-87f9-477a0fb7c10a).

Response


```
{
  "orderId": "0x53414e44424f58cd0982ee025144e5544139ce85f0efb93dee270145bafccb75",
  "route": "evm-btc-gasless",
  "state": "failed",
  "rawStatus": "SANDBOX_FAILED",
  "isComplete": true,
  "btcTxId": null,
  "txs": [],
  "failureReason": {
    "code": "solver_failed",
    "message": "The solver could not finish this order."
  },
  "sandbox": true,
  "scenario": "order_failed",
  "externalId": null,
  "externalCustomerId": null
}
```


`GET /orders/{id}`: a real sandbox answer, 13 November 2026 (`x-request-id` f6b14128-0369-46d9-8d81-e609a657f4e9).

10.8b `rawStatus` under each state

For support. Branch on `state`; `rawStatus` says where in it the order is.

`state`	`rawStatus` (EVM buys)	`rawStatus` (sells through the settler)	`rawStatus` (Solana)
`pending`	`PRE_ORDER`, `PENDING`, `ACCEPTED`, `PROCESSING`, `WATCHING`, `SUBMITTING_INTENT`, `INTENT_OPENED`, `DEFERRED`, `PERMIT2_COMMIT_PENDING`, `AUCTION`	`settler:NONE`	`OPENING`, `OPEN`, `TAKING`
`awaiting_deposit`		`settler:COMMITTED` (no payment seen yet)	
`in_progress`	`COMMITTED`, `COMMIT_UNVERIFIED`, `SENDING_BTC`, `PULLING_TOKENS`, `SWAPPING`, `TOKEN_SWAPPED`, `TOKEN_RECEIVED`, `BTC_CONFIRMED`, `SETTLING`, `BRIDGING`, `PROOF_READY`, `SETTLE_FAILED` (before the BTC is sent)	`settler:COMMITTED` (payment seen)	`FILLED`, `TAKEN`, `PAYING`, `LOCKED`, `PAID`, `SETTLING`
`completed`	`BTC_SENT`, `FILLED`, `COMPLETE`, `SETTLED`, or any of the above once `btcTxId` is set	`settler:FILLED`, `settler:SLASHED`	`SETTLED`, `FINALIZED`
`refundable`	`EXPIRED`, `FAILED`, `COMMIT_FAILED`		
`refunded`	`REFUNDED`, `SLASHED`		`REFUNDED`
`cancelled`	`CANCELLED`		`CANCELLED`, `COOPERATIVELY_CANCELLED`
`expired`		`settler:REFUNDED`, or the deposit window closed	`MISSED`, `EXPIRED`, `LATE_UNPAID`, `EXPIRED_AFTER_PAY`
`failed`			`FAILED`

10.8c Errors on this page

Code	HTTP	Cause	Fix
`invalid_input`	400	`orderId` is not an order id	Send the id your quote or order returned
`not_found`	404	No order has that transaction (`?txHash=`)	Check the hash; a new one may need a few seconds

Next: 10.9 Contracts.

Last verified: 13 November 2026, against the sandbox.

10.9 Contracts

After this page you can check every contract a Maxy order touches, on each chain's own explorer, and know who can change it and how.

Before you start: nothing. **Time:** 10 minutes.

Every address below is compiled into the SDK; a server response cannot add one or swap one for another. Compare the `to` of a real ETH escrow transaction from your tests with this table. A gasless escrow is sent by the solver's Safe, so its `to` is the Safe.

10.9a Settlers

The settler holds every escrow and releases it only against a Bitcoin payment proof from the verifier.

Chain	Chain id	Settler	Verifier	Explorer
Base	8453	`0xeC5a05b28dd848F628e19CB26Df32f053e64ab86`	`0x8dFED41f68b3749a06a0A9451E6d1701A9C75F9f`	basescan.org
Arbitrum	42161	`0x9e8d10Cae507ed849B140F0bC3C54e6b01067053`	`0xDbA3FaD120aE7bBBD42a39eAB65D2A41E030e29A`	arbiscan.io
Robinhood Chain	4663	`0x9e8d10Cae507ed849B140F0bC3C54e6b01067053`	`0xDbA3FaD120aE7bBBD42a39eAB65D2A41E030e29A`	robinhoodchain.blockscout.com
BNB Chain	56	`0xB0d873bF53b09A32139E45d225d16985227cA1Af`	`0xfE9F7A77AdB9BeC68E259c3016cb7154ded5acBc`	bscscan.com
Avalanche	43114	`0xB0d873bF53b09A32139E45d225d16985227cA1Af`	`0xfE9F7A77AdB9BeC68E259c3016cb7154ded5acBc`	snowtrace.io

Other contracts	Address	Role
Permit2	`0x000000000022D473030F116dDEE9F6B43aC78BA3` (every chain)	Moves USDC on the gasless route, bound to the order by a signed witness
Maxy's solver (a Safe)	`0x36E6B9FC60c90700359c7d4A44eb3fafb0837dA7` (every EVM chain)	Commits to orders, opens gasless escrows and fills sells. The BTC is paid from Maxy's Bitcoin wallet
Solana settlement program	`97CyZ9qdv7tec4LBM9d459mWLcZDv5CDcEH7Pw15QBHN`	Holds SOL escrows; with its Bitcoin light client `AVna5sHdDYF5uhS8KK7muNxCFe92nveopbHrfuq8Q12o` (solscan)

10.9b What is deployed

Read from each chain on the date below, and compared with `GET /chains`. The settler is a proxy, so its implementation and code hash are listed too. The first blocks are the deployment records of Maxy's own explorer, and are compared with the chain where a public RPC keeps that much history. `GET /pairs` names the same settlers.

Chain	Settler implementation	Implementation code hash	Verifier code hash	Settler first block	Verified source
Base	`0x3f603436…a74f`	`0xaab13741…541137`	`0x579a5789…fab149`	48598585	Sourcify
Arbitrum	`0xc3166fad…737f`	`0xd9be2e9a…3392fa`	`0x579a5789…fab149`	498275265	Sourcify
Robinhood Chain	`0xc3166fad…737f`	`0xd9be2e9a…3392fa`	`0x579a5789…fab149`	45120828	same code as the verified chains, apart from linked libraries
BNB Chain	`0x11450691…a759`	`0x151ea352…859059`	`0x579a5789…fab149`	109838048	same code as the verified chains, apart from linked libraries
Avalanche	`0x11450691…a759`	`0x151ea352…859059`	`0x579a5789…fab149`	90244340	same code as the verified chains, apart from linked libraries

The code hash is `keccak256` of the runtime bytecode (`eth_getCode`). The verifier's hash is the same on all five chains.

10.9c Source and versions

- The settler's source is verified on Sourcify for Base and Arbitrum (repo.sourcify.dev). The implementation on the other three chains is byte-for-byte the same code, apart from the addresses of four linked libraries and the metadata hash.
- The intent the user signs uses the EIP-712 domain `{ name: "BitcoinIntentSettler", version: "6", chainId, verifyingContract: <the settler> }`.
- The settler's ABI, with the functions these pages name (`openNativeBuyIntent`, `cancelBuyEscrow`, `slashExpired`, `refundExpired`, `fillIntentWithFeeToken`), are here as JSON: the fragments the SDK uses. The whole ABI comes with the verified source on Sourcify (Base, Arbitrum). The SDK carries its own copy and needs nothing from you.
- The verifier's bytecode is the same on all five chains.
- Audit status: audits.

10.9d Who can change what

Contract	Upgradeable	How
Settler	Yes: an EIP-1967 transparent proxy	Upgrades go through a timelock: 48 hours on Base, Arbitrum and Robinhood Chain; no delay on BNB Chain and Avalanche
Settler settings		A governance role can allow or remove bond tokens and settlement assets; changes to the bond split are timelocked 72 hours
Settler pause		A pauser role can stop new escrows and commits. Pausing never blocks cancel, settle, slash or refund, so no escrow can be trapped
Verifier	No: not a proxy	An admin can set the minimum confirmations (live: 1 on every chain) and its settings
Solana programs	Yes: upgradeable programs	One upgrade authority, `uzQY6w3CUEbRtZaaCUqMRMExAGmg5qL6L15nzC4JUNR`

10.9e What the settler guarantees

Read on the contract itself:

- An escrow leaves the settler only four ways: to the solver against a verified proof of the Bitcoin payment, or back to the user by cancel, slash or refund (security model).
- A solver can commit only before the intent's deadline (2 hours after signing).
- When a solver commits, anything the user escrowed above the settlement amount (the price band) goes straight back to the user.
- Cancel, slash and refund have no caller check after their deadline: anyone may send them, and the coins go only to the user (the bond split aside).

10.9f Errors on this page

None: this page reads chains, not the API. `GET /chains` takes your key and has no error of its own beyond `401` without one.

Next: 10.10 Chains and tokens.

Last verified: 4 November 2026, by reading each contract on chain.

10.10 Chains and tokens

After this page you know every chain and coin Maxy swaps, with addresses and decimals, and which way each pair runs.

Before you start: nothing. **Time:** 3 minutes.

The live lists are `GET /chains`, `GET /tokens?chain=…` and `GET /pairs` (with your key). This table is what they returned on the date below.

Chain	Chain id	CAIP-2	Buys BTC with	BTC sells into
Bitcoin	`bitcoin`	`bip122:000000000019d6689c085ae165831e93`		
Base	8453	`eip155:8453`	ETH (escrow), USDC (gasless)	ETH, USDC, USDT
Arbitrum	42161	`eip155:42161`	ETH (escrow), USDC (gasless)	ETH, USDC, USDT
BNB Chain	56	`eip155:56`	BNB (escrow)	BNB, USDC, USDT
Avalanche	43114	`eip155:43114`	AVAX (escrow)	AVAX, USDC, USDT
Robinhood Chain	4663	`eip155:4663`	ETH (escrow)	ETH, USDG
Solana	`solana`	`solana:5eykt4UsFv8P8NJdTREpY1vzqKqZKvdp`	SOL	SOL

A coin can also go by its CAIP-19 id wherever `from` and `to` take a name: `eip155:8453/erc20:0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913`, `eip155:8453/slip44:60`, `bip122:000000000019d6689c085ae165831e93/slip44:0`, `solana:5eykt4UsFv8P8NJdTREpY1vzqKqZKvdp/slip44:501`. Answers carry it as `assetId`.

`GET /pairs` also gives each pair's `route`, `settler`, smallest order (`minSats`, or `minUsd` where the solver prices it) and `requiredConfirmations`. Where `confirmationsBySize` is true, larger orders need more, and the firm quote states the number.

Response


```
{
  "pairs": [
    {
      "from": {
        "chain": 8453,
        "token": "0x0000000000000000000000000000000000000000",
        "symbol": "ETH",
        "decimals": 18,
        "chainName": "Base"
      },
      "to": {
        "chain": "bitcoin",
        "token": "BTC",
        "symbol": "BTC",
        "decimals": 8,
        "chainName": "Bitcoin"
      },
      "label": "ETH on Base → BTC",
      "fromAssetId": "eip155:8453/slip44:60",
      "toAssetId": "bip122:000000000019d6689c085ae165831e93/slip44:0",
      "route": "evm-btc-escrow",
      "settler": "0xeC5a05b28dd848F628e19CB26Df32f053e64ab86",
      "minSats": 546,
      "requiredConfirmations": 1,
      "confirmationsBySize": false
    },
    "…"
  ],
  "sandbox": true
}
```


`GET /pairs`: a real sandbox answer, 13 November 2026 (`x-request-id` d6dc7370-898b-4878-bd65-31a1022493b2), with long fields folded.

10.10a Read them

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/tokens?chain=base"
```

TypeScript

```
const { tokens } = await fetch(`${BASE}/tokens?chain=base`).then((r) => r.json());
```

Python

```
tokens = requests.get(f"{BASE}/tokens", params={"chain": "base"}).json()["tokens"]
```

Response

```
{
  "tokens": [
    {
      "chain": 8453,
      "chainName": "Base",
      "symbol": "ETH",
      "address": "0x0000000000000000000000000000000000000000",
      "decimals": 18,
      "assetId": "eip155:8453/slip44:60"
    },
    {
      "chain": 8453,
      "chainName": "Base",
      "symbol": "USDC",
      "address": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
      "decimals": 6,
      "assetId": "eip155:8453/erc20:0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913"
    },
    "…"
  ],
  "sandbox": true
}
```

`GET /tokens`: a real sandbox answer, 13 November 2026 (`x-request-id` b31d8fb7-6c7f-452c-a327-52658715851f), with long fields folded.

10.10b Tokens

Chain	Symbol	Address	Decimals
Bitcoin	BTC	`BTC`	8
Base	ETH	`0x0000000000000000000000000000000000000000`	18
Base	USDC	`0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913`	6
Base	USDT	`0xfde4C96c8593536E31F229EA8f37b2ADa2699bb2`	6
Arbitrum	ETH	`0x0000000000000000000000000000000000000000`	18
Arbitrum	USDC	`0xaf88d065e77c8cC2239327C5EDb3A432268e5831`	6

Chain	Symbol	Address	Decimals
Arbitrum	USDT	`0xFd086bC7CD5C481DCC9C85ebE478A1C0b69FCbb9`	6
BNB Chain	BNB	`0x0000000000000000000000000000000000000000`	18
BNB Chain	USDC	`0x8AC76a51cc950d9822D68b83fE1Ad97B32Cd580d`	**18**
BNB Chain	USDT	`0x55d398326f99059fF775485246999027B3197955`	**18**
Avalanche	AVAX	`0x0000000000000000000000000000000000000000`	18
Avalanche	USDC	`0xB97EF9Ef8734C71904D8002F8b6Bc66Dd9c48a6E`	6
Avalanche	USDt	`0x9702230A8Ea53601f5cD2dc00fDBc13d4dF4A8c7`	6
Robinhood Chain	ETH	`0x0000000000000000000000000000000000000000`	18
Robinhood Chain	USDG	`0x5fc5360D0400a0Fd4f2af552ADD042D716F1d168`	6
Solana	SOL	`SOL`	9

USDC and USDT on BNB Chain have 18 decimals, not 6: read `decimals` from the API rather than assuming it.

`GET /tokens` also lists a few tokens with `decimals: null` (such as WBTC and cbBTC). They have no pair yet.

10.10c Token list files

One token list per chain, written from `GET /tokens`, holding the tokens with known decimals: Base, Arbitrum, BNB Chain, Avalanche, Robinhood Chain.

10.10d Naming a coin

Every price and quote accepts the spelling your stack already uses: `USDC on base`, `arbitrum:USDC`, `BASE.ETH`, `USDC on eip155:8453`, a token address with `fromChain`, or `sellToken` with `chainId` (HTTP API).

10.10e Confirmations

Route	Bitcoin confirmations
Buying BTC (EVM)	1 before the solver can claim the escrow (the user has the BTC at once)

Route	Bitcoin confirmations
Selling BTC (EVM)	Set per order by the solver, 1 to 6; 1 on live orders
Solana routes	1 for early settlement

10.10f Errors on this page

Code	HTTP	Cause	Fix
`unsupported_chain`	400	A chain Maxy does not swap on	Pick from `GET /chains`

Next: 10.11 Limits.

Last verified: 13 November 2026, against the sandbox.

10.11 Limits

After this page you have every number that bounds an order, a call, a quote and a webhook, in one table.

Before you start: nothing. **Time:** 2 minutes.

`GET /limits` answers the numbers for your key:

curl

```
curl -s -H "x-maxy-key: $MAXY_API_KEY" "$BASE/limits"
```

TypeScript

```
const limits = await fetch(`${BASE}/limits`, { headers: { 'x-maxy-key': KEY } }).then((r) =>
r.json());
```

Python

```
limits = requests.get(f"{BASE}/limits", headers={"x-maxy-key": KEY}).json()
```

Response


```
{
  "kind": "secret",
  "scopes": [
    "read",
    "quote",
    "place",
    "orders",
    "webhooks",
    "account"
  ],
  "origins": [],
  "orderCapUsd": 100,
  "minOrderSats": 546,
  "routes": {
    "evm-btc-escrow": {
      "minSats": 546
    },
    "evm-btc-gasless": {
      "minSats": 546
    },
    "btc-evm": {
      "minUsd": 10
    },
    "sol-btc": {
      "minUsd": 2
    },
    "btc-sol": {
      "minUsd": 2
    }
  },
  "quoteTtlSeconds": 60,
  "batchMax": 20,
  "rateLimit": {
    "limit": 120,
    "remaining": 118,
    "resetSeconds": 60,
    "per": "key",
    "priceLimit": 1200
  },
  "sandbox": true
}
```


`GET /limits`: a real sandbox answer, 13 November 2026 (`x-request-id` 2a9d335f-ceff-43fc-9c3d-5b741f61a402).

Without a key: `401 auth_required`, as on every route but `/health` and the status page.

What	Value	Source
Smallest BTC buy	546 sats after costs	`minOrderSats`, `limits.minSats` on each buy
Smallest BTC sell to an EVM chain	about $10	sell BTC
Largest order	Your key's order size cap, $100 unless agreed	`orderCapUsd`
Open sell orders	3 per wallet	sell BTC
Calls, live key	600 a minute per key	`rateLimit`
Calls, test key	120 a minute per key; 50 open orders	`rateLimit`
Price reads	Ten times the above, counted apart; the same question within 3 seconds is one read	`rateLimit.priceLimit`
Quotes in one batch	20	`batchMax`
Streams	Two minutes a connection, a read every 5 seconds or more	streams
Smallest order per route	`routes` in `GET /limits`, and `minSats` or `minUsd` on each pair	`routes`
Quote lifetime	60 seconds	`quoteTtlSeconds`
A solver must commit	Within 2 hours of signing	the intent's `deadline`
A solver must pay	Within 144 Bitcoin blocks of its commit, about a day	`settlementTimeoutBlocks`
A sell's deposit window	Live orders: 19 Bitcoin blocks, about 3 hours	`order.settler.btcDeadline`
Webhook endpoints	5 per key	`409 conflict` above it
Webhook retries	9 attempts over about 23 hours	webhook events
Orders listed	100 a page with a cursor, 1,000 in a CSV; kept 30 days after a final state	reconcile
Sandbox orders kept	7 days	sandbox

Largest sizes per route are not published: a size Maxy cannot fill is refused after the click, with no figure (order size and limits).

10.11a Errors on this page

Code	HTTP	Cause	Fix
`invalid_api_key`	401	A malformed or unknown key	Check it
`rate_limited`	429	Over the limit	Wait `Retry-After` seconds

Next: 10.12 Runnable samples.

Last verified: 13 November 2026, against the sandbox.

10.12 Runnable samples

After this page you have a runnable file for every step of the buy, with the real response it returned.

Before you start: an API key; curl and jq, Node 22.18+ or Python 3.10+. **Time:** 2 minutes to run the first one.

The samples ship in your partner kit, in `examples/`. A local script runs them against the sandbox and saves the exact output beside each one, with the `x-request-id`, rate-limit and timing headers. The files are `NN-step.sh.out.txt`, `NN-step.ts.out.json` and `NN-step.py.out.json`. `last-run.json` holds the date and the versions of that run: it is tested with the SDK below, Node and Python as listed there.

Nothing is sent and no funds move. Wherever a signature is needed, a sample makes a throwaway key at run time. Bitcoin goes to the BIP-173 test vector address `bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4`: never send real BTC to it. No file holds a key; every sample reads `MAXY_API_KEY`.

10.12a Run one

curl

```
export MAXY_API_KEY=mxk_test_…
./02-price.sh
```

TypeScript

```
// from the kit folder: mkdir maxy-samples && cd maxy-samples && npm init -y && npm pkg set
type=module
// npm install https://maxy.co/sdk/maxy-labs-sdk-latest.tgz viem && cp ../examples/*.ts
../examples/webhook-test-vector.json . && node 02-price.ts
```

Python

```
# python3 -m venv .venv && .venv/bin/pip install -r requirements.txt
# .venv/bin/python 02-price.py
```

10.12b The files

Step	Files	What it does
1	`01-pairs.{sh,ts,py}`	`GET /pairs`
2	`02-price.{sh,ts,py}`	`GET /price` for 10 USDC on Base
3	`03-quote.{sh,ts,py}`	`GET /quote`: `quoteId`, `expiresAt`, `issues`, `steps`
4	`04-sign.{sh,ts,py}`	Sign each `signature` step's typed data (cast, viem, eth_account), checking it hashes to `step.hash`
5	`05-order.{sh,ts,py}`	Quote, sign, `POST /orders`: the escrow transaction to send, and its simulation
6	`06-status.{sh,ts,py}`	`GET /status` by order id and by transaction hash, for a real completed order
7	`07-webhooks.{sh,ts,py}`	Register a webhook, send a test event, list events, delete it, verify the test vector
8	`08-orders.{sh,ts,py}`	`GET /orders` as JSON and as CSV
9	`09-key.{sh,ts,py}`	`GET /key`: your key and its order size cap
	`quickstart.{ts,py}`	The first script: a price, a quote for an order, the steps. About 10 seconds from a fresh install
	`webhook-test-vector.json`, `verify-webhook.mjs`, `verify_webhook.py`	A signed delivery and its verification in Node and Python: accepted, refused with one byte changed, refused 301 seconds late
	`errors/`	Each error answer the guide names, captured live
	`first-order.{ts,py}`	A whole order with the flow helper, and with the Python client
	`sandbox.{sh,ts,py}`	One whole order to `completed` with a test key
	`Maxy.postman_collection.json`	Every route, for Postman, written from `GET /openapi.json`
	`captures/`	Every route's real sandbox answer, with its `x-request-id`: what this guide's response blocks are filled from
	`paths/adapter.ts`	Aggregator path: discovery, a route, a skipped size, tracking by hash. Runs against the sandbox
	`paths/webhook-receiver.mjs`	Embed path: verifies each delivery, processes each event `id` once. `--self-test` runs it on the test vector
	`paths/worker.py`	Backend path: quote, sign, order, poll, export, from a server. Runs against the sandbox
	`paths/next-route.ts`, `paths/buy-btc.ts`	Frontend path: copy into a Next.js app with wagmi. Not runnable on their own

Steps 3 to 5 were captured with `FROM="ETH on base" AMOUNT=0.004`. Live, a USDC order's steps come only for a wallet that holds the USDC. A throwaway holds none, so its live quote answers `issues.balance` and `steps: null`. The same code signs every step of kind `signature`, keyed by the step's id.

10.12c The first script

TypeScript

```
// The first script: a price, then the firm quote for an order, then the steps the user's wallet
would sign and send.
// Nothing is signed or sent. From a clean machine (Node 22.18 or later runs .ts directly;
older: npx tsx quickstart.ts):
//
//   mkdir maxy-quickstart && cd maxy-quickstart && npm init -y && npm pkg set type=module
//   npm install https://maxy.co/sdk/maxy-labs-sdk-latest.tgz viem
//   cp /path/to/quickstart.ts . && MAXY_API_KEY=mxk_live_… node quickstart.ts
//
// FROM (default "USDC on base"), AMOUNT in whole coins (default 10), FROM_ADDRESS (default: a
throwaway wallet with
// no funds), TO_ADDRESS (default: the BIP-173 test vector address; never send real BTC to it).
import { createMaxyClient } from '@maxy-labs/sdk';
import { generatePrivateKey, privateKeyToAccount } from 'viem/accounts';

const BASE = process.env.MAXY_API_BASE ??
'https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api';
const KEY = process.env.MAXY_API_KEY;
if (!KEY) throw new Error('export MAXY_API_KEY=mxk_live_…');
const from = process.env.FROM ?? 'USDC on base';
const amount = process.env.AMOUNT ?? '10';
const units = (raw: string, decimals: number) => (Number(raw) / 10 **
decimals).toLocaleString('en-US', { maximumFractionDigits: decimals });

// 1. A price with the SDK: what the amount gets right now. No wallet needed.
const maxy = createMaxyClient({ apiKey: KEY });
const price = await maxy.swap.price({ from, to: 'BTC', amount });
console.log(`price   ${price.formatted?.send} → ${price.formatted?.receive}
(${price.route})`);

// 2. The firm quote for the order, over HTTP: who pays, who receives, and the steps.
const fromAddress = process.env.FROM_ADDRESS ??
privateKeyToAccount(generatePrivateKey()).address;
const toAddress = process.env.TO_ADDRESS ?? 'bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4';
const res = await fetch(`${BASE}/quote?${new URLSearchParams({ from, to: 'BTC', amount,
fromAddress, toAddress })}`, { headers: { 'x-maxy-key': KEY } });
const quote = await res.json();
if (!res.ok) {
  console.log(`quote   ${res.status} ${quote.error.code}: ${quote.error.message}   (request
${quote.requestId})`);
  process.exit(1);
}
console.log(`quote   ${quote.formatted.send} → ${quote.formatted.receive}   from ${fromAddress}
to ${toAddress}`);
console.log(`        quoteId ${quote.quoteId ?? '(none)'}   expires ${new Date(quote.expiresAt *
1000).toISOString()}   request ${res.headers.get('x-request-id')}`);
```

```
// 3. What is in the way, and the steps.
const { balance, orderCap } = quote.issues ?? {};
if (orderCap) console.log(`issue   ${orderCap.message}`);
if (balance) console.log(`issue   the wallet holds ${units(balance.actual, quote.from.decimals)}
${quote.from.symbol}; the order needs ${units(balance.expected, quote.from.decimals)}
${quote.from.symbol}`);
if (!quote.steps) {
  console.log('steps   none yet: a USDC order is signed (permit, witness) by a wallet that holds
the USDC, so its steps come');
  console.log("        for such a wallet (set FROM_ADDRESS). FROM='ETH on base' AMOUNT=0.004
shows the escrow steps for any wallet.");
} else {
  quote.steps.forEach((s: { id: string; kind: string; description: string; hash?: string; path?:
string }, i: number) => {
    console.log(`step ${i + 1}  ${s.id.padEnd(8)} ${s.kind.padEnd(12)} ${s.description}`);
    if (s.hash) console.log(`        ${''.padEnd(21)} EIP-712 digest ${s.hash}`);
  });
  console.log('        nothing signed or sent: 04-sign.ts and 05-order.ts take it from here');
}
```

Python

```
"""The first script: a price, then the firm quote for an order, then the steps the user's wallet
would sign and send.

Nothing is signed or sent. From a clean machine (Python 3.10 or later):

    python3 -m venv .venv && .venv/bin/pip install requests eth-account
    MAXY_API_KEY=mxk_live_… .venv/bin/python quickstart.py

FROM (default "USDC on base"), AMOUNT in whole coins (default 10), FROM_ADDRESS (default: a
throwaway wallet with no
funds), TO_ADDRESS (default: the BIP-173 test vector address; never send real BTC to it).
"""
import os
import sys
from datetime import datetime, timezone

import requests
from eth_account import Account

BASE = os.environ.get("MAXY_API_BASE",
"https://whhxyutitykcsdpvdgcd.supabase.co/functions/v1/maxy-api")
KEY = os.environ.get("MAXY_API_KEY") or sys.exit("export MAXY_API_KEY=mxk_live_…")
FROM = os.environ.get("FROM", "USDC on base")
AMOUNT = os.environ.get("AMOUNT", "10")


def units(raw: str, decimals: int) -> str:
    return f"{int(raw) / 10 ** decimals:,.{decimals}f}".rstrip("0").rstrip(".")


# 1. A price: what the amount gets right now. No key or wallet needed.
price = requests.get(f"{BASE}/price", params={"from": FROM, "to": "BTC", "amount": AMOUNT},
timeout=60).json()
print(f"price   {price['formatted']['send']} → {price['formatted']['receive']}
({price['route']})")

# 2. The firm quote for the order, with your key: who pays, who receives, and the steps.
from_address = os.environ.get("FROM_ADDRESS") or Account.create().address
to_address = os.environ.get("TO_ADDRESS", "bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4")
r = requests.get(f"{BASE}/quote", headers={"x-maxy-key": KEY}, timeout=60, params={
    "from": FROM, "to": "BTC", "amount": AMOUNT, "fromAddress": from_address, "toAddress":
to_address,
})
quote = r.json()
if not r.ok:
    sys.exit(f"quote   {r.status_code} {quote['error']['code']}: {quote['error']['message']}
(request {quote['requestId']})")
```

```
expires = datetime.fromtimestamp(quote["expiresAt"], timezone.utc).isoformat().replace("+00:00",
"Z")
print(f"quote   {quote['formatted']['send']} → {quote['formatted']['receive']}   from
{from_address} to {to_address}")
print(f"        quoteId {quote['quoteId'] or '(none)'}   expires {expires}   request
{r.headers.get('x-request-id')}")

# 3. What is in the way, and the steps.
issues = quote.get("issues") or {}
if issues.get("orderCap"):
    print(f"issue   {issues['orderCap']['message']}")
if issues.get("balance"):
    b, sym, dec = issues["balance"], quote["from"]["symbol"], quote["from"]["decimals"]
    print(f"issue   the wallet holds {units(b['actual'], dec)} {sym}; the order needs
{units(b['expected'], dec)} {sym}")
if not quote.get("steps"):
    print("steps   none yet: a USDC order is signed (permit, witness) by a wallet that holds the
USDC, so its steps come")
    print("        for such a wallet (set FROM_ADDRESS). FROM='ETH on base' AMOUNT=0.004 shows
the escrow steps for any wallet.")
else:
    for i, s in enumerate(quote["steps"], 1):
        print(f"step {i}  {s['id']:<8} {s['kind']:<12} {s['description']}")
        if s.get("hash"):
            print(f"        {'':<21} EIP-712 digest {s['hash']}")
    print("        nothing signed or sent: 04-sign.py and 05-order.py take it from here")
```

What it printed:

```
$ node quickstart.ts
price   10.00 USDC on Base → 0.00011999 BTC   (evm-btc-gasless)
quote   10.00 USDC on Base → 0.00011999 BTC   from 0x6F8f8a559C2ea48bb837e960419534f446d82DFc to
bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
        quoteId 0x53414e44424f5812e5dba3586ab7ef6d4733bbc40474963afd6cf4e27382de5b   expires
2026-11-04T18:48:38.000Z   request 781ccb89-d0d9-4f73-842b-a73f160126af
step 1  witness  signature    Sign the transfer bound to this order. No gas. Sandbox: this
signature authorizes nothing.
                              EIP-712 digest
0xd6d0cbc31700fb46c80732956ca49496962adf392ee2e9b02af348e6a0fa57d4
step 2  order    post         Send the order and its signatures. The solver opens the escrow and
pays the gas.
        nothing signed or sent: 04-sign.ts and 05-order.ts take it from here

$ FROM='ETH on base' AMOUNT=0.004 node quickstart.ts
price   0.004 ETH on Base → 0.000116 BTC   (evm-btc-escrow)
quote   0.004 ETH on Base → 0.000116 BTC   from 0xe7e17aE4191D7544d4eeC9fb36458Bcd8EE7e244 to
bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
        quoteId 0x53414e44424f58ba4c62b43e0f91df4394f7ddcd96be397d6b78e52f2dddc111   expires
2026-11-04T18:48:38.000Z   request 5e19d8e1-9202-4291-a739-897b6662fcba
step 1  intent   signature    Sign the order (EIP-712). No gas. Sandbox: this signature
authorizes nothing.
                              EIP-712 digest
0x2f2bbff600ec27bebca8babfd7c7f10764f8c73c38de1ce6a65c24c984feb04d
step 2  order    post         Send the order and its signature; you get back the transaction to
send.
step 3  escrow   transaction  Sandbox: do not send the transaction. POST /orders/submit with its
simulatedHash.
        nothing signed or sent: 04-sign.ts and 05-order.ts take it from here
```

What it printed with a test key, 4 November 2026.

10.12d Errors on this page

A sample prints the API's own error and exits. The codes are on error codes.

Next: 10.13 FAQ.

Last verified: 5 November 2026, against the sandbox.

10.13 FAQ

This page is for evaluators and integrators with a specific question; each answer links to the page that covers it in full.

Does the SDK hold keys or custody funds? No. Every signature and transaction goes through a signer you pass in (signers). The SDK builds, checks and broadcasts; it never sees a private key.

How do I get the SDK and an API key? Both are provided during partner onboarding, together with the registration of your sign-in domains (getting started).

What is the API key? Your integration's own key (`mxk_live_…`). It identifies your integration to the Maxy API, so Maxy can count your calls, cap your orders and revoke the key on its own. It asks for quotes, places orders your users sign, reads the orders placed with it and registers webhooks. It cannot move funds or sign, and grants no access to any wallet's records; calls that act for a wallet also need that wallet's session token (keys and limits).

Can I run a solver, or send orders to a solver directly? Solving is open: any solver that posts the bond can commit to an order on the settler, and the escrow pays it only against a proof of the Bitcoin payment (operations). Orders placed through the SDK go through the Maxy API, which relays them to Maxy's solver; there is no other order channel in the SDK. Solana routes are the exception: they are authorized by the wallet's own signatures and go to the Solana settlement service (buy BTC from Solana).

Which Bitcoin addresses can receive BTC? EVM to BTC: native SegWit only (`bc1q...`, P2WPKH or P2WSH). SOL to BTC also accepts Taproot (`bc1p...`). Paying from the SDK (`payDeposit`, MaxyLock funding, Maxy Mint deposits) needs a P2WPKH or P2SH-P2WPKH account with its public key. The table per route is in swap.

Why did `createOrder` throw `verification_failed` when nothing looked wrong? The SDK compares what servers and wallets return with what it computed itself (the intent hash against the settler's, the signer recovered from each signature, the signed Bitcoin transaction against the built one, the LI.FI calldata, a MaxyLock plan against the quote). Any difference stops the flow before money moves. Read `error.message` and `error.serverCode`; retrying with fresh data usually works if a quote moved, but a repeated mismatch is worth reporting.

My Bitcoin broadcast returned `ambiguous`. Should I pay again? No. No node confirmed it, but it may have reached the network. Look up the txid first.

How do I get an order back after a page reload? Keep what `createOrder` returned (or at least `{ route, orderId, chainId }`) and call `swap.getStatus`. For lists use `orders.list({ wallet, tracked, mintOrderIds })` (orders).

When can I refund an EVM to BTC escrow? An escrow nobody committed to: after its deadline, 2 hours after you signed. A committed order the solver did not pay: after the Bitcoin deadline (about 144 blocks, roughly a day). `swap.planRefund` tells you which, and when (refunds and failures).

Does the SDK log anything? No. Every BTC deposit address is the settler escrow's own address, read from the contract (security, guarantee 13), so there is nothing to warn about.

Is there a sandbox or a testnet? There is a sandbox for swaps: a test key (`mxk_test_…`) runs the same API, SDK and CLI with nothing real behind them, and can force every outcome (sandbox). There is no testnet for swaps: the last step before launch is a few small mainnet orders (testing and going live).

Where is the service status? `GET /health` (no key) gives the solver, Bitcoin and each chain's verifier, with a `degraded` flag; there is no status page (support and status).

How do I report a bug or a vulnerability? Report a bug through your onboarding channel with the SDK version, the call, the `MaxyError` `code`, `serverCode` and `message`, and any order id or transaction hash. Report a vulnerability as described in 27.4 Security policy.

What is the difference between a price and a quote? A price is for display: no wallet, nothing reserved. A quote is for an order about to be placed: it needs both addresses, and holds for 60 seconds. Both take your key (pricing anatomy).

How long does a quote last? 60 seconds. `POST /orders` answers `409 quote_expired` after that (firm quotes and expiry).

What is the largest order I can place? Your key's order size cap, $100 unless agreed otherwise. `GET /key` shows it (order size and limits).

What is the smallest order? 546 sats after costs for a BTC buy, and about $10 for a BTC sell (limits).

How long until the user has the BTC? Maxy's solver typically sends it within 3 seconds of committing. The user's wallet shows it after the first confirmation, about 10 minutes (how a swap settles).

Does the user need gas to buy with USDC? No. On Base and Arbitrum the wallet only signs, and the solver pays the gas (gasless buy).

Which Solana wallets work? Wallets that inject the provider shape Phantom, Solflare and Backpack use (signers).

Is there a retry-safe way to post an order? Yes: post it again. The order id is the idempotency key (retries).

How do I find an order from a transaction hash? `GET /status?txHash=` takes the escrow transaction or the Bitcoin txid (status by transaction hash).

How do I verify a webhook? HMAC-SHA256 of `<t>.<raw body>` with the endpoint secret, or `verifyWebhook` in the SDK. A test vector is on webhook events.

What happens if my endpoint is down? Maxy retries 8 more times over about 23 hours. After that, read `GET /status` for each open order (delivery).

How do I export orders for month-end? `GET /orders?from=&to=&format=csv` (reconcile and export orders).

What happens if a sell is paid the wrong amount? The contract cannot prove it, so the payout cannot be released. Contact Maxy support with the txid (refunds and failures).

Who pays the user back when an order fails? Nobody needs to: the user's wallet takes the coins back from the settler (refunds and failures).

Can Maxy move or freeze an escrow? No. It leaves the settler only by proof, cancel, slash or refund, and pausing never blocks the last three (security model).

Has anyone outside Maxy reviewed the contracts? No independent audit has been completed. 27.2 Audits and assurance lists each component and what exists today.

Does Maxy perform KYC or screen addresses? No. Your own rules apply in your product (partner obligations, screening).

What data does Maxy store? Per order, the signed intent, the signatures, the payout script and the status. Per call, the operation, time, IP address and API key (data handling).

Which languages are supported? TypeScript has an SDK. Any other language calls the HTTP API; every guide shows curl and Python (10.2 API reference).

Is the SDK on npm? Not on the public registry. npm installs it from maxy.co: `npm install https://maxy.co/sdk/maxy-labs-sdk-latest.tgz viem`; Python: `pip install https://maxy.co/sdk/maxy-latest.tar.gz`. Checksums: https://maxy.co/sdk/SHA256SUMS.

How will I hear about a breaking change? At least 90 days ahead, to every key holder (versioning and deprecation).

10.13a Where next

- Documentation index
- 25.2 Getting started
- Errors
- Changelog feed

Next: 10.14 Changelog.

Last verified: 7 November 2026, against the SDK source and the API it runs.

10.15 Migration guides

After this page you know which breaking changes have happened, what replaced them, and how a future one will reach you.

Before you start: nothing. **Time:** 2 minutes.

10.15a SDK and API 0.15.0: an order ends when the user's funds arrive

5 November 2026. The API keeps its shape: no field changed its name.

Before: a Solana buy read `in_progress` until `rawStatus` was `SETTLED`.

TypeScript

```
if (status.rawStatus === 'SETTLED') showSuccess(status.btcTxId);
```

After: it reads `completed` when the BTC is sent, the same as an EVM buy. A settlement that runs late or fails no longer changes what the user sees.

TypeScript

```
if (status.state === 'completed') showSuccess(status.btcTxId);
```

Before: a publishable key answered `scope_denied` on `POST /orders`. After: it may place the user's own signed order, so a page can run the whole flow. If you relied on that refusal, keep orders on your server with `placeOrder`.

10.15b The HTTP API

The API has had no breaking change. `GET /orders` now answers a page of up to 100, where it answered up to 1,000 in one call: pass `limit` and `cursor`, or use `format=csv` for up to 1,000. `/health` keeps `status` and adds parts. When a breaking change comes, it comes as `/v2` with its own guide here, written before the notice goes out (versioning and deprecation).

10.15c The SDK

The SDK is below 1.0, so a breaking change raises the minor version. These are the ones that change code you would write:

Version	Before	After
0.14.0	A secret key in a browser page worked	It throws at construction. Use a publishable key there, or pass `dangerouslyAllowBrowser: true` knowing anyone can read it
0.14.0	`MaxyError` was the only class	Every failure is a subclass, so `error.name` changed. `instanceof MaxyError`, `isMaxyError` and `error.code` work as before
0.13.4	`swap.getSellTemplate()` returned a sizing field	The field is gone. A size the solver cannot fill is refused with `insufficient_liquidity`
0.11.0	Separate size-check calls on `swap`	Removed. Call `swap.quote()` and read `issues.orderCap` and `liquidityAvailable`

TypeScript

```
// ask for the order, and read what the quote says
const quote = await maxy.swap.quote({ from: 'USDC on base', to: 'BTC', amount: '150',
fromAddress, toAddress });
if (quote.issues?.orderCap) console.log(quote.issues.orderCap.message);
```

Every other change is in the changelog.

10.15d Errors on this page

None: this page has no call.

Next: 11.1 How awards work.

Last verified: 5 November 2026, against the SDK source and the API it runs.

11.1 How awards work

After this page you can explain the partner program's epochs and formula. The full terms are at maxy.co/learn/partners, and they win over this summary.

Before you start: nothing. **Time:** 2 minutes.

11.1a The summary

Term	Value
Pool	10% of Maxy's fully diluted equity, issued as shares
Calendar	30 epochs of 36 days, 1,080 days in all. The start date is on the terms page
Pool per epoch	Epochs 1 to 10: 0.50% of Maxy each. Epochs 11 to 20: 0.30%. Epochs 21 to 30: 0.20%
Your share	Your settled BTC ÷ all partners' settled BTC × that epoch's pool
To earn	Settle at least 1 BTC in the epoch. Below that the epoch earns no award, and nothing else changes
What counts	Orders of at least 0.001 BTC that settled against a Bitcoin proof. Quotes, cancels and refunds do not
When	The award is issued in full when the epoch closes

11.1b A worked epoch

Partners settle 400 BTC in epoch 3, and 100 of it is yours. You hold 25% of a 0.50% pool, which is 0.125% of Maxy.

11.1c Your records

`GET /orders` lists the orders placed with your key, for your own count (reconcile and export orders).

11.1d Errors on this page

None: this page has no call.

Next: 12.1 Source adapter.

Last verified: 5 November 2026, against the SDK source and the API it runs.

12.1 Source adapter

After this page you have the reading order for adding Maxy to your router, the finish line, and three links to forward.

Before you start: 1.1 What Maxy is. **Time:** 15 minutes to read the list; the pages take about an hour.

You are: an aggregator or router engineer wrapping Maxy as one more source.

12.1a Pages, in order

1. 1.1 What Maxy is
2. 3.2 Pricing anatomy
3. 3.4 Order size and limits
4. 3.5 Firm quotes and expiry
5. 4.2 Aggregator or router adapter
6. 10.1 HTTP API: pairs, quote and status by hash
7. 10.6 Error codes
8. 10.11 Limits
9. 8.3 Test scenarios: over cap and expired quote
10. 8.5 Go-live checklist

12.1b Done when

Your adapter maps a Maxy quote to your route schema and reads `/pairs` for discovery. It skips Maxy when a quote carries `issues.orderCap` or `liquidityAvailable: false`, instead of failing at execution. It tracks orders by `txHash`, and `GET /health` reading `degraded` takes Maxy out of your ranking.

12.1c What to send your team

1. The path: 4.2 Aggregator or router adapter
2. The reference implementation: 10.12 Runnable samples, `examples/paths/adapter.ts`
3. The checklist: 8.5 Go-live checklist

12.1d Errors on this page

None: this page has no call. Each page above lists its own.

Next: 12.2 Product embed.

Last verified: 5 November 2026, against the SDK source and the API it runs.

12.2 Product embed

After this page you have the reading order for a BTC buy in your product, the finish line, and three links to forward.

Before you start: 1.1 What Maxy is. **Time:** 30 minutes to read the list; the pages take a few hours.

You are: a wallet, exchange or on-ramp engineer putting a BTC buy inside an existing product.

12.2a Pages, in order

1. 2.1 Quickstart
2. 3.1 Order lifecycle
3. 3.6 Security model
4. 3.7 Environments and keys
5. 4.3 Wallet, exchange or on-ramp embed
6. 5.1 Buy BTC from an EVM chain
7. 5.3 Gasless buy
8. 5.4 Buy BTC from Solana
9. 5.5 Sell BTC to a chain
10. 5.7 Track orders and receive webhooks
11. 5.10 Refunds and failures
12. 5.9 Reconcile and export orders
13. 6.1 Showing a quote and the UX pages after it
14. 7.1 Keys and secrets and the security pages after it
15. 9.1 Status and maintenance
16. 8.5 Go-live checklist

12.2b Done when

A user buys BTC from the wallet they already have, and the BTC lands in that wallet. Every state and failure shows approved words. Webhooks update your ledger once per event, and support has the refunds page.

12.2c What to send your team

1. The path: 4.3 Wallet, exchange or on-ramp embed
2. The reference implementation: 10.12 Runnable samples, `examples/paths/webhook-receiver.mjs`
3. The checklist: 8.5 Go-live checklist

12.2d Errors on this page

None: this page has no call. Each page above lists its own.

Next: 12.3 Frontend button.

Last verified: 5 November 2026, against the SDK source and the API it runs.

12.3 Frontend button

After this page you have the reading order for a Buy BTC button, the finish line, and three links to forward.

Before you start: 1.1 What Maxy is. **Time:** 10 minutes to read the list; the pages take about an hour.

You are: a DeFi or DEX frontend engineer adding Buy BTC on the wallet connection the dapp already has.

12.3a Pages, in order

1. 2.1 Quickstart
2. 4.4 DeFi or DEX frontend
3. 6.1 Showing a quote and the UX pages after it
4. 7.1 Keys and secrets
5. 7.3 Content Security Policy
6. 8.5 Go-live checklist

12.3b Done when

A Buy BTC entry point ships on your existing wagmi or wallet connection. Your secret key stays on your server, a publishable key bound to your origin reads prices and quotes in the page, and the API host is in your Content Security Policy.

12.3c What to send your team

1. The path: 4.4 DeFi or DEX frontend
2. The reference implementation: 10.12 Runnable samples, `examples/paths/next-route.ts` and `buy-btc.ts`
3. The checklist: 8.5 Go-live checklist

12.3d Errors on this page

None: this page has no call. Each page above lists its own.

Next: 12.4 Headless backend.

Last verified: 5 November 2026, against the SDK source and the API it runs.

12.4 Headless backend

After this page you have the reading order for signing orders server-side, the finish line, and three links to forward.

Before you start: 1.1 What Maxy is. **Time:** 15 minutes to read the list; the pages take about an hour.

You are: a backend or custodial platform engineer signing on your own servers.

12.4a Pages, in order

1. 2.1 Quickstart
2. 4.5 Backend-only
3. 5.1 Buy BTC from an EVM chain
4. 3.8 Rate limits, retries and idempotency
5. 5.7 Track orders and receive webhooks
6. 5.9 Reconcile and export orders
7. 10.11 Limits
8. 9.1 Status and maintenance
9. 9.3 Support and escalation
10. 8.5 Go-live checklist

12.4b Done when

Orders are created and signed server-side with your own key management. Every order call is safe to retry. Month-end reconciles from `GET /orders` without asking Maxy.

12.4c What to send your team

1. The path: 4.5 Backend-only
2. The reference implementation: 10.12 Runnable samples, `examples/paths/worker.py`
3. The checklist: 8.5 Go-live checklist

12.4d Errors on this page

None: this page has no call. Each page above lists its own.

Next: 12.5 Vendor review.

Last verified: 5 November 2026, against the SDK source and the API it runs.

12.5 Vendor review

After this page you have the reading order for reviewing Maxy as a vendor, the finish line, and three links to forward.

Before you start: 1.1 What Maxy is. **Time:** 30 minutes.

You are: a security, compliance or procurement reviewer deciding whether Maxy can be a vendor.

12.5a Pages, in order

1. 1.1 What Maxy is
2. 3.6 Security model
3. 7.5 What Maxy does and does not do
4. 7.6 Partner obligations
5. 7.7 Screening and decline codes
6. 10.9 Contracts
7. 27.2 Audits and assurance
8. 27.3 Data handling
9. 27.4 Security policy
10. 9.1 Status and maintenance
11. 7.8 Reporting an incident
12. What does not exist

12.5b Done when

Each technical question on your questionnaire is answered by a link above, or by the "what does not exist" list. Ask your Maxy contact for anything left over.

12.5c What to send your team

1. The statement: 7.5 What Maxy does and does not do
2. The contracts: 10.9 Contracts
3. The policy: 27.4 Security policy

12.5d Errors on this page

None: this page has no call. Each page above lists its own.

Next: the track you started from, or 1.4 Which path is mine.

Last verified: 5 November 2026, against the SDK source and the API it runs.