Maxy integration guide: the short version

Published by Maxy. 12 November 2026. Each sample prints in one language; the full guide has curl, TypeScript and Python. Numbers are the full guide's.

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"
```

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"
```

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.

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
```

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)"
```

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
`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);
```

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.

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"
```

`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
```

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.

TypeScript

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

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

To test	Scenario	What happens
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"
```

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.