KodaCash API integration guide

API v1 · Merchant integration · ARS QR payment walkthrough

Resource	Address
API base URL	https://kodacash.app/integrations
Partner portal	https://kodacash.app/cabinet/
English PDF	https://kodacash.app/docs/partner-api-en.pdf
English plain text	https://kodacash.app/docs/partner-api-en.txt
OpenAPI specification	https://kodacash.app/integrations/openapi.json

Sign in and issue an API key

Sign in to the partner portal with the credentials issued by KodaCash. Open Connection (Подключение), select your API merchant and copy its merchant_id.

Select Issue key (Выпустить ключ) and confirm. In the dialog, select Copy (Скопировать) and save the secret in your secrets manager. Once you close the dialog, the portal shows only the key ID. Send the secret in the Authorization: Bearer <API_KEY> header.

If an active key already exists, use its saved secret. To replace it, select Reissue key (Перевыпустить ключ). Confirming immediately revokes the old key. If the issue button is unavailable, ask your administrator for key issuance permission.

Required permissions

API operation	Permission
GET /v1/capabilities	catalog.read
POST /v1/payment-checks	checks.create, checks.read
GET /v1/payment-checks/{check_id}	checks.read
POST /v1/payment-checks/{check_id}/refresh	checks.refresh, checks.read
POST /v1/payment-checks/{check_id}/confirm	checks.confirm, checks.read
GET /v1/orders/{order_id}	orders.read
GET /v1/orders/{order_id}/events	orders.read

KodaCash assigns permissions and scope. Each key is bound to a merchant_id and to its approved modes and routes. For this ARS QR example, live execution requires separate live access, an enabled route, a merchant automation rule and limits.

Create a sandbox payment check

These examples use Bash, curl and jq 1.7 or later. In your Bash session, run read -rs API_KEY, enter the secret and press Enter. The input is hidden. Replace your-merchant-id in check.json with your actual merchant_id.

1. Check whether the route is available

```
BASE=https://kodacash.app/integrations
curl -sS -i -H "Authorization: Bearer $API_KEY" \
  "$BASE/v1/capabilities?mode=sandbox"
```

The portal calls sandbox mode “Симуляция” and live mode “Боевой”. In the HTTP 200 response, look in routes for currency="ARS" and payment_method="qr". Read amount_scale and qr_formats for that route. An empty routes array means this key has no routes in the selected mode.

2. Save the request body as check.json

```
{
  "merchant_id": "your-merchant-id",
  "external_order_id": "ars-order-001",
  "idempotency_key": "ars-check-001",
  "currency": "ARS",
  "amount": "3000.00",
  "payment_method": "qr",
  "payment_data": {
    "qr_payload": "KODA-SANDBOX:ARS:success"
  },
  "execution_mode": "sandbox"
}
```

For each new operation, use a new external_order_id and idempotency_key. Each must be 1-128 characters and match ^[A-Za-z0-9][A-Za-z0-9._:-]{0,127}$. Send amount as a positive decimal string with a decimal point and up to two fractional digits for ARS. Preserve the full precision of rates and amounts in pricing and settlement using a decimal type.

3. Submit the check

```
curl -sS -o check-response.json -w '%{http_code}\n' \
  -H "Authorization: Bearer $API_KEY" -H 'Content-Type: application/json' \
  --data-binary @check.json "$BASE/v1/payment-checks"
```

HTTP 201 means the check was saved. The response body in check-response.json contains the check object. Review the terms once check.status is "ready".

Review the sandbox quote

4. Read the current terms

```
jq '.check | {check_id,status,can_confirm,expires_at,order_id,
  amount,currency,payee,payee_verified,pricing,quote_fingerprint,
  failure_code,limits}' check-response.json
```

Verify amount, currency and payee. You can confirm when status is "ready", can_confirm is true and the current expires_at (ISO 8601) is in the future. An initial rate estimate may expire before the overall payment check.

For an initial estimate, payee may come from the QR or be null. If payee_verified is false, the provider has not yet verified the recipient. When the provider returns its own terms, use the payee in that response. Those terms may omit payee_verified.

Selected fields from a simulated 3,000 ARS check appear below. Always take check_id and quote_fingerprint from your own response.


```
{
  "check_id": "53906c53-65de-4c5d-b5bb-bf7a8d7c2364",
  "amount": "3000.00",
  "currency": "ARS",
  "status": "ready",
  "can_confirm": true,
  "payee_verified": false,
  "rate_observed_at": "2026-11-19T12:28:12Z",
  "quote_fingerprint": "1765222c07869e7e636c915c128620e9ab0ad1c07411a6b7db5c0d4c6343b59e",
  "pricing": {
    "estimated_amount": "1.75",
    "estimated_rate": "0.000583107379223884",
    "koda_fee_amount": "0",
    "koda_fee_currency": "USDT",
    "rate": "0.000583107379223884",
    "rate_base": "ARS",
    "rate_quote": "USDT",
    "rate_status": "estimated",
    "version": 0
  }
}
```


Read or refresh an existing check

GET /v1/payment-checks/{check_id} returns the saved check. While it is valid, POST an empty JSON object {} to /v1/payment-checks/{check_id}/refresh to update its terms. Review the terms and quote_fingerprint again after a refresh. If status is "failed", read failure_code.

Confirm the sandbox payment

5. Confirm the approved terms

```
CHECK_ID=$(jq -ber '.check.check_id' check-response.json)
FINGERPRINT=$(jq -ber '.check.quote_fingerprint' check-response.json)
jq -n --arg f "$FINGERPRINT" '{confirm:true,quote_fingerprint:$f}' > confirm.json
curl -sS -o confirm-response.json -w '%{http_code}\n' \
  -H "Authorization: Bearer $API_KEY" -H 'Content-Type: application/json' \
  --data-binary @confirm.json "$BASE/v1/payment-checks/$CHECK_ID/confirm"
```

HTTP 202 with order and check objects means the order was accepted. Save order.order_id. The final debit can exceed the USDT estimate approved by the customer.

If the API returns HTTP 409 quote_changed

```
{
  "error": {
    "code": "quote_changed"
  }
}
```

A new recipient or an increase over the initial estimate requires fresh customer approval. If the recipient is unchanged and the provider estimate is at or below the current estimate, the first confirmation can return HTTP 202 immediately.

After quote_changed, read the same check_id. If it already has an order_id, poll that order.

```
curl -sS -o check-response.json -w '%{http_code}\n' \
  -H "Authorization: Bearer $API_KEY" "$BASE/v1/payment-checks/$CHECK_ID"
```

With HTTP 200 and order_id=null, review the recipient, amount, pricing, expires_at and can_confirm again. After the customer accepts, repeat step 5 using the new quote_fingerprint from GET. If it is unchanged, check failure_code and can_confirm. If needed, refresh a still-valid check and review the resulting terms.

In the 3,000 ARS example, pricing.initial_estimated_amount retained the original 1.75 USDT estimate. The updated provider estimate in pricing.estimated_amount was 1.80 USDT.

Retrieve the result and final debit

6. Poll the order until settlement values are available

After HTTP 202, take the order ID from confirm-response.json. If you recovered an existing order via the check instead, use check.order_id.

```
ORDER_ID=$(jq -ber '.order.order_id' confirm-response.json)
curl -sS -o order.json -w '%{http_code}\n' \
  -H "Authorization: Bearer $API_KEY" "$BASE/v1/orders/$ORDER_ID"
jq '{order_id,status,status_version,pricing,settlement}' order.json
```

Even after status becomes "succeeded", keep polling the same order_id until the fields you need are present. In live mode, background reconciliation fills these fields.

Selected fields from the 3,000 ARS simulation follow. The final rate was 0.00061 USDT per ARS and the simulated debit was 1.83 USDT.


```
{
  "order_id": "53906c53-65de-4c5d-b5bb-bf7a8d7c2364",
  "currency": "ARS",
  "amount": "3000.00",
  "status": "succeeded",
  "execution_mode": "simulated",
  "pricing": {
    "estimated_amount": "1.80",
    "estimated_rate": "0.00060",
    "final_rate_source": "provider",
    "initial_estimated_amount": "1.75",
    "initial_rate": "0.000583107379223884",
    "koda_fee_amount": "0",
    "koda_fee_currency": "USDT",
    "rate": "0.00061",
    "rate_base": "ARS",
    "rate_quote": "USDT",
    "rate_status": "confirmed",
    "source": "provider",
    "version": 0
  },
  "settlement": {
    "actual_debit": "1.83",
    "asset": "USDT",
    "estimated_amount": "1.80",
    "fee_amount": null,
    "fee_currency": null,
    "rate": "0.00061",
    "rate_base": "ARS",
    "rate_quote": "USDT",
    "status": "confirmed"
  }
}
```


Read the final rate from pricing.rate when pricing.rate_status is "confirmed". Read the debit from settlement.actual_debit, denominated in settlement.asset. These values can arrive in different responses. A field can still be null even when settlement.status is "confirmed".

In this example, pricing.koda_fee_amount is zero. Where provided, settlement.fee_amount and settlement.fee_currency report the provider fee.

Handle retries and move to live

Order statuses and events

Status	Action
accepted / processing	Continue polling the original order.
pending_reconciliation	The outcome is still being checked. Keep the original order_id and continue polling.
succeeded	The payment was executed. Check the latest settlement fields.
failed / expired	The order failed definitively or expired before dispatch. Read failure_code.

GET /v1/orders/{order_id}/events returns HTTP 200 with an events array. Each order snapshot includes status_version. Apply only newer versions and deduplicate events by event_id. Poll the order and its events every 5 seconds initially, backing off to 15-30 seconds while waiting.

Retries and errors

Situation	Action
Retry of POST /v1/payment-checks	Reuse the body and idempotency_key. A retry returns the same check_id and HTTP 201. A different body with the same key returns HTTP 409 idempotency_conflict.
Timeout or 5xx after confirm	Read the original check. If it has an order_id, poll that order. Otherwise retry the original confirmation with the same fingerprint. An exact retry of an accepted confirmation returns the same order.
check_expired	First check the original order_id. If no order exists, start a new attempt with a new external_order_id and idempotency_key.

Read error.code or check.failure_code. For HTTP 401, check your key. For 403, verify permissions, merchant_id and mode. For 404, verify the ID and access scope. For 409, inspect can_confirm, expires_at, failure_code and limits. For rate_policy_changed, refresh a still-valid check.

Live execution

After separate live approval, call GET /v1/capabilities?mode=live. For this ARS route, submit a new check with fresh external_order_id and idempotency_key values, execution_mode="live" and the full payload of a valid Argentine QR. HTTP 202 after confirm means the order was accepted for real execution. A direct POST /v1/orders in live mode returns HTTP 409 preparation_required.

For recognized fixed-amount QR codes, the API compares the QR amount with amount. If the QR omits an amount, send the agreed amount. Mismatches can produce qr_amount_mismatch or provider_qr_amount_mismatch. QR limits are 4,096 UTF-8 bytes in live mode and 8,192 in sandbox. The JSON limit is 16,384 bytes. The provider may impose additional QR limits.