

- [Document Control](#)
- [1. Quick start](#)
 - [1.1 What you are building](#)
 - [1.2 One session, end to end](#)
 - [1.3 Six steps to go-live](#)
- [2. Common to every call](#)
 - [2.1 Headers and signature: GA → your wallet](#)
 - [2.2 Worked example](#)
 - [2.3 Signature: your calls to GA](#)
 - [2.4 The wallet envelope](#)
 - [2.5 Two answers](#)
 - [2.6 Money](#)
 - [2.7 Timeouts and retries](#)
 - [2.8 Errors on GA's side, in one place](#)
- [3. Idempotency: the four rules that protect money](#)
 - [Rule 1. A repeated `op\_id` gets the same answer](#)
 - [Rule 2. Check things in this order](#)
 - [Rule 3. Never refuse a win, a rollback, or a grant settlement for session, block, or limits](#)
 - [Rule 4. A rollback of something you never saw is a success, and you remember it](#)
- [4. You call GA: launch, catalog, free rounds, reports](#)
 - [4.1 Launch a game](#)
 - [4.2 Launch a demo](#)
 - [4.3 Catalog](#)
 - [4.4 Free rounds you issue](#)
 - [4.5 Sessions, rounds, and reports](#)
 - [4.6 Event stream \(optional\)](#)
- [5. GA calls your wallet: the actions](#)
 - [5.1 `get\_balance`](#)
 - [5.2 `authenticate` \(optional\)](#)
 - [5.3 `debit`](#)
 - [5.4 `credit`](#)
 - [5.5 `debit\_credit` \(optional\)](#)
 - [5.6 `rollback`](#)
 - [5.7 `close\_round` \(optional\)](#)
 - [5.8 `reconcile`](#)
 - [5.9 `settle\_grant` \(mandatory if you issue free rounds\)](#)
 - [5.10 `notify` \(optional\)](#)
- [6. Optional: provider features](#)
 - [6.1 The calls](#)
- [7. Data formats](#)
- [8. Network and limits](#)
- [9. Certification and go-live](#)
 - [9.1 Run the checks from the Operator Portal](#)
 - [9.1a The command-line tool \(for your CI\)](#)
 - [9.2 The checklist](#)
 - [9.3 Changelog](#)
- [Appendix A. Every error code](#)
 - [A.1 Codes your wallet returns to GA](#)
 - [A.2 What GA returns to you](#)
- [Appendix B. Numbers to remember](#)

Document Control

Item	Value
Contract version	2.0.0 (frozen, additive-only)
Document revision	2026-09-25
Exact field types	<code>proto/*.proto</code> and <code>openapi/*.yaml</code> in this package. Where this guide and those files differ, the files govern; tell us at integration@game-alligator.com
Money rules in full	<code>money-path/OPERATOR-API-MONEY-PATH-RULES.md</code> (the Money Path Rules page on the docs site)
Postman	

Item	Value
	postman/GA_Operator_API_v2 (your wallet), postman/GA_Aggregator_API_v2 and postman/GA_Features_API_v2 (calls to GA), environment postman/GA_Sandbox_v2
Certification	Operator Portal → Integration → Callback test, or the ga-operator-certify tool for CI; sign-off form certification/GA_Operator_Certification_Checklist_v2.md (chapter 9)
Support	integration@game-alligator.com

1. Quick start

1.1 What you are building

Game Alligators (GA) puts game studios' games into your casino. Traffic runs in two directions, each with its own key pair. Don't mix them.

Direction	Who calls whom	What it's for	Key you use	Chapter
You → GA	You call <code>https://api.rexplay.site/v2/...</code>	Launch a game, read the catalog, issue free rounds, read reports	Operator API key + hash-based message authentication code (HMAC) secret, from your onboarding profile	4
GA → you	GA calls your wallet at <code>https://<your-host>/v2/wallet/*</code>	Balance, bets, wins, rollbacks	Wallet signing key, from the Operator Portal. GA signs, you verify	5

Transports: HTTP/JSON, which this guide covers, or native gRPC with the same messages. On gRPC there is no HMAC signature in either direction: the credential travels as metadata (`x-api-key` plus `ga-brand-id` on calls to your wallet, `x-api-key` plus `x-brand-id` on your calls to GA). If you already run a Softgaming-compatible wallet API, ask GA about the compatibility adapter. A separate document covers it.

1.2 One session, end to end

Player	Your site	GA	Your wallet
click game			
----->	launch_game		
	----->	(token, player, currency)	
	<-----	launch_url, session 24h	
<-----	open launch_url		
		get_balance / authenticate	
		----->	
		<-----	balance
spin		debit (op_id A)	
		----->	
		<-----	ok, balance_after
		credit (op_id B, win)	
		----->	
		<-----	ok
spin, no answer		debit (op_id C) ... 5 s per attempt, 8 s per bet	
		----->	(timeout)
		rollback (original C)	
		----->	
		<-----	ok, original_found: false
		(late debit C arrives -> you refuse ALREADY_ROLLED_BACK)	

1.3 Six steps to go-live

1. Get sandbox credentials. Email `integration@game-alligator.com`. You receive an operator profile with `operator_id`, the Operator API key pair, and a login to `https://operator.rexplay.site`. There you rotate the wallet signing key once and copy the secret. The portal shows it only at that moment.
2. Give GA your sandbox wallet URL and the optional wallet actions you support. The URL uses HTTPS and is reachable from the internet. Also give your currencies with their exponent, a 24/7 technical contact, and a finance contact.
3. Build the five mandatory wallet actions (chapter 5) with the four money rules (chapter 3). Test your signature check with the worked example in §2.2 first.

4. Run the Postman collection `GA_Operator_API_v2` against your wallet. Set `wallet_base_url`, `ga_wallet_key_id`, `ga_wallet_hmac_secret`. Done when all ten requests answer a valid envelope.
5. Run the certification checks from the Operator Portal (chapter 9) until every check passes, then sign the checklist with your GA integration manager.
6. Repeat steps 2, 4, and 5 with production credentials and your production wallet URL. Production credentials are separate. GA never derives them from sandbox ones.

Addresses:

Surface	Sandbox	Production
GA API (chapter 4)	<code>https://api.rexplay.site</code>	<code>https://api.game-alligator.com</code>
Operator Portal (wallet key, reports)	<code>https://operator.rexplay.site</code>	<code>https://operator.game-alligator.com</code>
Your wallet	you host it, you give GA the URL	separate URL and credentials
GA calls your wallet from	<code>49.13.169.177</code> , <code>46.224.156.1</code>	<code>49.13.169.177</code>

2. Common to every call

2.1 Headers and signature: GA → your wallet

Every call GA sends to your wallet looks like this:

```
POST /v2/wallet/debit HTTP/1.1
Host: wallet.operator.example
Content-Type: application/json
X-API-Key-Id: key-live-01
X-Request-Id: 0198a1d0-9a30-7f08-a7dd-713e4fd33db0
Idempotency-Key: op-bet-48912
X-GA-Signature: t=1783944000,v1=8a75f608c2a8e178d89a055198050e9a2d5341c2f390ac1d2c2c68e433eb5f2d
```

- `X-API-Key-Id` names the wallet signing key. It's the key id you see in the Operator Portal.
- `X-Request-Id` is this one network attempt. It equals `request_id` in the body and `payload.meta.request_id`, and changes on every retry.
- `Idempotency-Key` equals `payload.meta.op_id`, the business operation. It stays the same across retries. Use it, or the body field, as your idempotency key.
- `X-GA-Signature` carries the timestamp `t` (unix seconds, the same instant as the body's `ts`) and the signature `v1`.

Verify in this order, before you parse the body:

1. Find your secret by `X-API-Key-Id`.
2. Split `X-GA-Signature` into `t` and `v1`.
3. Reject if `t` is more than 300 seconds away from your clock.
4. Compute `SHA256_HEX` of the raw body bytes exactly as received. Don't re-serialize.
5. Compute `HMAC-SHA256(secret, t + "." + request_id + "." + body_sha256)` where `request_id` is the top-level `request_id` field of the body (identical to the `X-Request-Id` header). Compare with `v1` using a constant-time compare.
6. Check that `operator_id` in the body is your operator id.

If any step fails, answer HTTP 401 or 403 with no body. GA treats that as “no answer” and retries. A bet that never verifies ends in a rollback.

2.2 Worked example

Use these values to test your verification code before GA sends anything. Secret, body, and timestamp are fixed, so your output must match to the byte.

Input	Value
Wallet HMAC secret	sec-cert-01
<code>t</code>	1783944000 (the body's <code>ts</code> 2026-07-13T12:00:00Z)
<code>request_id</code>	0198a1d0-9a30-7f08-a7dd-713e4fd33db0
Raw body (one line, no trailing newline)	<pre>{"request_id":"0198a1d0-9a30-7f08-a7dd-713e4fd33db0","ts":"2026-07-13T12:00:00Z","operator_id":"0197aaaa-0000-7000-a000-000000000002","action":"get_balance","payload":{"meta":{"request_id":"0198a1d0-9a30-7f08-a7dd-713e4fd33db0","op_id":"op-bal-1001"},"operator_id":"0197aaaa-0000-7000-</pre>

Input	Value
	a000-000000000002", "player_ref": "player-10428"}, "currency": "EUR"}}
SHA256_HEX(body)	f6ebf9ad22209d716b34fff30cb66d7232e96a2d403e86ed0dc23e5da9f8ad6b
Signing input	1783944000.0198a1d0-9a30-7f08-a7dd-713e4fd33db0.f6ebf9ad22209d716b34fff30cb66d7232e96a2d403e86ed0dc23e5da9f8ad6b
Expected v1	8a75f608c2a8e178d89a055198050e9a2d5341c2f390ac1d2c2c68e433eb5f2d

Node.js:

```
const crypto = require("crypto");
function verify(rawBody, headers, secret) {
  const m = /t=(\d+),v1=([0-9a-f]{64})/.exec(headers["x-ga-signature"] || "");
  if (!m) return false;
  const [, t, v1] = m;
  if (Math.abs(Date.now() / 1000 - Number(t)) > 300) return false;
  const requestId = JSON.parse(rawBody).request_id;
  const bodySha = crypto.createHash("sha256").update(rawBody).digest("hex");
  const expected = crypto.createHmac("sha256", secret).update(`${t}.${requestId}.${bodySha}`).digest();
  return crypto.timingSafeEqual(expected, Buffer.from(v1, "hex"));
}
```

PHP:

```
function verify(string $rawBody, array $headers, string $secret): bool {
    if (!preg_match('/t=(\d+),v1=([0-9a-f]{64})/', $headers['X-GA-Signature'] ?? '', $m)) return false;
    [, $t, $v1] = $m;
    if (abs(time() - (int)$t) > 300) return false;
    $requestId = json_decode($rawBody, true)['request_id'] ?? '';
    $input = $t . '.' . $requestId . '.' . hash('sha256', $rawBody);
    return hash_equals(hash_hmac('sha256', $input, $secret), $v1);
}
```

Go:

```
func verify(rawBody []byte, sigHeader, secret string) bool {
    var t int64; var v1 string
    if _, err := fmt.Sscanf(sigHeader, "t=%d,v1=%s", &t, &v1); err != nil { return false }
    if d := time.Now().Unix() - t; d > 300 || d < -300 { return false }
    var env struct{ RequestID string `json:"request_id"` }
    _ = json.Unmarshal(rawBody, &env)
    sum := sha256.Sum256(rawBody)
    mac := hmac.New(sha256.New, []byte(secret))
    mac.Write([]byte(fmt.Sprintf("%d.%s.%s", t, env.RequestID, hex.EncodeToString(sum[:]))))
    want, _ := hex.DecodeString(v1)
    return hmac.Equal(mac.Sum(nil), want)
}
```

Python:

```
import hashlib, hmac, json, re, time
def verify(raw_body: bytes, sig_header: str, secret: str) -> bool:
    m = re.fullmatch(r"t=(\d+),v1=([0-9a-f]{64})", sig_header or "")
    if not m: return False
    t, v1 = m.groups()
    if abs(time.time() - int(t)) > 300: return False
    request_id = json.loads(raw_body)["request_id"]
    signing_input = f"{t}.{request_id}.{hashlib.sha256(raw_body).hexdigest()}"
    expected = hmac.new(secret.encode(), signing_input.encode(), hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, v1)
```

Java:

```

static boolean verify(byte[] rawBody, String sigHeader, String secret) throws Exception {
    var m = java.util.regex.Pattern.compile("t=(\\d+),v1=([0-9a-f]{64})").matcher(sigHeader == null ? "" :
        sigHeader);
    if (!m.matches()) return false;
    long t = Long.parseLong(m.group(1)); String v1 = m.group(2);
    if (Math.abs(System.currentTimeMillis() / 1000 - t) > 300) return false;
    String requestId = new
        com.fasterxml.jackson.databind.ObjectMapper().readTree(rawBody).get("request_id").asText();
    String bodySha =
        java.util.HexFormat.of().formatHex(java.security.MessageDigest.getInstance("SHA-256").digest(rawBody));
    var mac = javax.crypto.Mac.getInstance("HmacSHA256");
    mac.init(new javax.crypto.spec.SecretKeySpec(secret.getBytes(java.nio.charset.StandardCharsets.UTF_8),
        "HmacSHA256"));
    byte[] expected = mac.doFinal((t + "." + requestId + "." +
        bodySha).getBytes(java.nio.charset.StandardCharsets.UTF_8));
    return java.security.MessageDigest.isEqual(expected, java.util.HexFormat.of().parseHex(v1));
}

```

2.3 Signature: your calls to GA

Same header names (X-API-Key-Id, X-GA-Signature: t=...,v1=...), your Operator API key pair, the same 300-second window, but a different signing input:

```

SIGNING_INPUT = METHOD + "\n" + PATH + "\n" + t + "\n" + SHA256_HEX(raw_body)
v1            = HMAC-SHA256(operator_api_secret, SIGNING_INPUT)

```

METHOD is POST, PATH is the URL path only (/v2/aggregator/launch_game), t is unix seconds. The Postman collections GA_Aggregator_API_v2 and GA_Features_API_v2 compute this for you. Set ga_api_key_id and ga_hmac_secret.

Use these values to test your signing code before you call GA. Secret, body, and timestamp are fixed, so your output must match to the byte.

Input	Value
Operator API HMAC secret	sec-op-cert-01
METHOD	POST
PATH	/v2/aggregator/launch_game
t	1783944000
Raw body (one line, no trailing newline)	{"player_ref":"player-10428","game_id":"pragmatic-vs20olympgate","currency":"EUR","token":"lt-cert-556677","return_url":"https://operator.example/lobby"}
SHA256_HEX(body)	fd1f9b4dff076382033df7e7f9e575f62cebefb6cdc28276fc6e95ec4f411eb0
Signing input	POST\n/v2/aggregator/ launch_game\n1783944000\nfd1f9b4dff076382033df7e7f9e575f62cebefb6cdc28276fc6e95ec4f411eb0
Expected v1	d1de5557c4a004e53eec6e34930e331638497c15884d17e2a649a75ce82d5283

The header you send is:

```

X-API-Key-Id: key-cert-op-01
X-GA-Signature: t=1783944000,v1=d1de5557c4a004e53eec6e34930e331638497c15884d17e2a649a75ce82d5283

```

Note the separator. Unlike §2.1's GA→wallet signature (which joins t, request_id, and the body hash with .), the signing input here joins METHOD, PATH, t, and the body hash with newlines (\n). There is no trailing newline after the hash.

2.4 The wallet envelope

Every wallet request has the same outer shape. `payload` is the action-specific part and the per-call ids live in `payload.meta`.

```
{
  "request_id": "0198a1d0-9a30-7f08-a7dd-713e4fd33db0",
  "ts": "2026-09-13T12:00:00Z",
  "operator_id": "0197aaaa-0000-7000-a000-000000000002",
  "action": "debit",
  "correlation": { "provider_slug": "pragmatic", "provider_round": "round-82721", "provider_ref": "spin-99182" },
  "payload": {
    "meta": {
      "request_id": "0198a1d0-9a30-7f08-a7dd-713e4fd33db0",
      "op_id": "op-bet-48912",
      "operator_id": "0197aaaa-0000-7000-a000-000000000002",
      "session_id": "0198a1cb-4f37-7ee8-8e7d-08d39925aec0",
      "player_ref": "player-10428",
      "launch_token": "lt-abcdef123456",
      "provider_slug": "pragmatic",
      "game_code": "vs20olympgate",
      "round_id": "round-82721",
      "provider_ref": "spin-99182",
      "ts": "2026-09-13T12:00:00Z"
    },
    "money": { "currency": "EUR", "amount": 100, "exponent": 2 }
  }
}
```

Field	Meaning
op_id	The business operation. The same bet has the same op_id on every retry. Store your answer under it.
request_id	This one network attempt. Changes on every retry. Logs only.
round_id	Groups the operations of one game round. Never an idempotency key.
session_id, launch_token	The player session and the token you gave at launch. Absent on session-less credits, such as promo payouts.
player_ref	Your player id, passed through unchanged.
correlation	The game provider's own ids, for support tickets.

Ignore fields you don't know. New optional fields may appear. GA never removes or renames anything published.

2.5 Two answers

Success:

```
{
  "status": "ok",
  "data": {
    "operator_tx_id": "tx-op-982103",
    "balance_after": { "currency": "EUR", "amount": 14900, "exponent": 2 },
    "debited": { "currency": "EUR", "amount": 100, "exponent": 2 }
  }
}
```

Refusal, always HTTP 200, the reason in the body, no balance:

```
{
  "status": "error",
  "error": {
    "code": "ERROR_CODE_INSUFFICIENT_FUNDS",
    "message": "Player available balance is below requested debit amount",
    "retryable": false
  }
}
```

You may add `retry_after` (a duration such as `"5s"`) to a retryable code. `attributes` is a free-form string map for your own diagnostics. GA doesn't read it.

2.6 Money

```
{ "currency": "EUR", "amount": 100, "exponent": 2 }
```

- `amount` is an integer in minor units. `100` with `exponent: 2` is 1.00 EUR. `exponent` is 0 to 18. GA never sends less than 2.
- Take the exponent from the message, never from your own currency table. If it differs from what you hold the player's currency in, refuse with `ERROR_CODE_CURRENCY_MISMATCH`.
- On your wallet `amount` is a JSON number: `100`. On GA's own APIs (chapter 4) it's a JSON string: `"100"`, because proto3 JSON encodes every 64-bit integer as a string. Same value, same minor units. A parser that accepts both is the safe choice.

2.7 Timeouts and retries

The most important number: GA waits 5 seconds for each wallet call. Answer faster than that, or GA treats the call as lost.

GA sees only two kinds of outcome:

- Refused. A valid envelope with a non-retryable code, such as `INSUFFICIENT_FUNDS`. Final: GA doesn't resend that `op_id`.
- No answer. A timeout, a dropped connection, a bare HTTP status with no envelope, or a valid envelope with a retryable code (`RATE_LIMITED`, `MAINTENANCE`, `INTERNAL`). GA doesn't know whether you applied the operation and retries with the same `op_id` and a new `request_id`.

Operation	On "no answer" GA does
<code>debit</code> , <code>debit_credit</code>	Up to 4 attempts, 100 ms, 500 ms, and 2 s apart, all inside one 8-second budget for the bet. Each attempt waits at most 5 s or what's left of the budget, whichever is shorter. GA starts a retry only if the budget still holds the pause before it. So a wallet that doesn't answer at all gets 2 attempts (5 s, then the rest of the 8 s). All 4 happen only when your wallet answers fast with a retryable code. Still nothing: the bet fails towards the game and GA sends one rollback for that <code>op_id</code> .
<code>credit</code> , <code>settle_grant</code>	The same quick attempts, then a background resend with the same <code>op_id</code> for up to 72 hours. Backoff starts at 1 second and doubles up to a cap of 10 minutes. The limit is the 72 hours, not a number of attempts. GA never rolls back a win.
<code>rollback</code>	The same quick attempts, then the same 72-hour resend. Stops at the first valid envelope: success or refusal.
<code>notify</code>	One attempt, no retry on HTTP/JSON. On gRPC it gets the same quick attempts as every other call.

GA honours a `Retry-After` header or `retry_after` field, capped at 10 seconds inside the quick attempts. When the 72 hours are over, GA flags the operation for manual review and alerts its team. GA never drops anything silently.

What this means for your wallet:

1. A bet can reach you after GA already rolled it back. Rule 4 (chapter 3) handles it.
2. A win can reach you days later, after the session ended. Rule 3 handles it.
3. Every retry carries the same `op_id`, so Rule 1 makes retries harmless.

2.8 Errors on GA's side, in one place

Your wallet's codes are Appendix A.1. What GA returns to you is Appendix A.2. Every error from `/v2/aggregator/*` and `/v2/features/*` has a non-200 HTTP status and one JSON body, `Content-Type: application/json`:

```
{"code": "ERROR_CODE_MAINTENANCE", "message": "provider_unavailable", "retryable": true}
```

`code` is one of the 15 `ERROR_CODE_*` values of Appendix A.1. `message` is for your logs and GA support. It starts with a stable token (`signature_invalid`, `game_unavailable`, `provider_timeout`, ...). GA omits `retryable` when false. This holds for signature and key problems (401, 403) and for a read-only key on a write call (403) too.

Retry only 429, 503, and 504 (`retryable: true` in the body), with backoff. Every other status, 502 included, is final for that request.

3. Idempotency: the four rules that protect money

Certification tests these hardest. Build them in from day one. The full normative text is in Money Path Rules.

Rule 1. A repeated `op_id` gets the same answer

Keep every `op_id` with its stored answer for at least 4 months. GA resends for 72 hours, and the margin covers reconciliation disputes.

- Same `op_id`, same content: return the stored answer, move no money. Even if the balance is now too low or the session expired.
- Same `op_id`, different content (player, operation type, currency, amount with exponent, round, original, grant): refuse with `ERROR_CODE_IDEMPOTENCY_CONFLICT`, move no money.
- Two identical requests at the same moment: the second waits for the first and gets the same answer. Never apply twice.

Rule 2. Check things in this order

```
signature -> schema -> op_id lookup -> player + currency/exponent  
-> [bets only: session, player status, limits, balance]  
-> round state -> apply and store the answer in one atomic step
```

The `op_id` lookup comes before session, player, and balance checks. Otherwise a repeated bet answers “insufficient funds” or “session expired” where it must answer the stored success, and you and GA disagree about whether the bet exists.

Rule 3. Never refuse a win, a rollback, or a grant settlement for session, block, or limits

A win can arrive hours after the session ended, for a blocked player, or past a deposit limit. Accept it. The only final refusals for a credit are: player not found, currency mismatch, invalid request, idempotency conflict, unknown original, already rolled back. After any other refusal, GA resends the credit for 72 hours, and then it lands in manual review.

A promo, bonus, jackpot, or tournament payout may arrive with no bet in the round and with no session. Don’t require either.

Rule 4. A rollback of something you never saw is a success, and you remember it

If a rollback names an `original_op_id` you don’t have:

1. Return `status: ok` with `"original_found": false`. Move no money.
2. Store that `original_op_id` as “rolled back”.
3. If the original debit or credit arrives later, refuse it with `ERROR_CODE_ALREADY_ROLLED_BACK`.

Never return `ERROR_CODE_UNKNOWN_ORIGINAL` to a rollback. Step 2 is what stops you from charging a player for a bet GA already cancelled.

4. You call GA: launch, catalog, free rounds, reports

Base URL `https://api.rexplay.site` (sandbox) or `https://api.game-alligator.com` (production). Every call is POST with a JSON body and the signature of §2.3. Paths are `/v2/aggregator/<call>` in snake_case. The sandbox also accepts the proto method name (`/v2/aggregator/LaunchGame`, case-sensitive). Sign exactly the path you call. Success is 200 with the call's own response. Amounts are strings in minor units (§2.6). Timestamps are Request for Comments (RFC) 3339. Paginated calls take `page: {cursor, limit}` and return `page: {next_cursor, has_more, total_count}`. `limit` defaults to 50, with a maximum of 500.

Start from Postman: import `postman/GA_Aggregator_API_v2.postman_collection.json` with `postman/GA_Sandbox_v2.postman_environment.json`, set `ga_api_key_id` and `ga_hmac_secret`, and send `launch_game` first. The pack with both files is `/downloads/GA_Operator_Integration_Pack_v2.0.0.zip` on the docs site.

4.1 Launch a game

POST `/v2/aggregator/launch_game`

Field	Required	Meaning
<code>player_ref</code>	yes	Your player id. Opaque to GA, returned unchanged in every wallet call.
<code>game_id</code>	yes	From the catalog (§4.3).
<code>currency</code>	yes	The currency of this session. Uppercase code. GA takes the exponent from your onboarding profile.
<code>token</code>	yes	Your launch token. GA stores it and sends it back as <code>launch_token</code> in every wallet call of this session. It's how you tie a wallet call to a player session.
<code>return_url</code>	no	Where the game's "back to lobby" button leads.
<code>attributes</code>	no	String map. <code>attributes["language"]</code> = two-letter lowercase code ("de"). Anything else falls back to "en".

Response: `session_id`, `launch_url`, `expires_at` (the session lives 24 hours), plus running `total_bet`, `total_win`, `rounds`.

`launch_game` is idempotent by `token` while the session is open. The same token with the same player, game, and currency returns the first response again: the same `session_id`, `launch_url`, and `expires_at`. So a page reload doesn't open a second session, and GA doesn't call the game provider again.

GA refuses the same token with 409 `ERROR_CODE_IDEMPOTENCY_CONFLICT` (`session_token_conflict`) in these cases:

- the token comes with another player, game, or currency;
- the session of that token is closed: by `close_session`, or by a newer launch of the same game for the same player, because a launch closes that player's earlier sessions of the game;
- the session of that token is past its `expires_at`.

GA never reopens a closed session. Don't retry the 409: launch again with a fresh token. If the first launch with the token is still running, GA answers 503 `ERROR_CODE_INTERNAL` with `retryable: true`

(`launch_in_progress`). Retry with the same token to get the first response. Use a fresh token for every new session.

Open `launch_url` in an `iframe` or by redirect. Both work. The URL is single-session: launch again for a new session. Device, IP, and country aren't fields on this call.

If you embed `launch_url` in an `iframe`, set a `sandbox` attribute. Don't omit it, and don't use an unrestricted frame. `launch_url` always points to a real address that you load via `src`, never as inline HTML, so the minimal working set is:

```
<iframe
  src="https://play.rexplay.site/s/0198a1cb-4f37-7ee8-8e7d-08d39925aec0"
  sandbox="allow-scripts allow-same-origin allow-forms allow-popups"
  allow="autoplay; fullscreen"
  allowfullscreen>
</iframe>
```

- `allow-scripts`—the game is a script-driven client. Without it, nothing runs.
- `allow-same-origin`—the frame loads from `src`, so this keeps it same-origin with GA's domain, not yours, which the game's own websocket connection and storage need. It doesn't weaken your page's origin boundary.
- `allow-forms`—some game and payment flows submit forms inside the frame.
- `allow-popups`—rule and payment/cashier windows some games open are new browser contexts, not just navigation inside the frame.
- `allow="autoplay; fullscreen"` and `allowfullscreen`—background/video-driven games and fullscreen mode need these. Playback or fullscreen silently fails without them.

```
{ "player_ref": "player-10428", "game_id": "0198a2aa-1111-7000-a000-000000000077",
  "currency": "EUR", "token": "lt-abcdef123456", "return_url": "https://casino.example/lobby",
  "attributes": { "language": "de" } }
```

```
{ "session_id": "0198a1cb-4f37-7ee8-8e7d-08d39925aec0", "player_ref": "player-10428",
  "game_id": "0198a2aa-1111-7000-a000-000000000077", "currency": "EUR",
  "launch_url": "https://play.rexplay.site/s/0198a1cb-4f37-7ee8-8e7d-08d39925aec0",
  "expires_at": "2026-09-14T12:00:00Z", "total_bet": { "currency": "EUR", "amount": "0", "exponent": 2 },
  "total_win": { "currency": "EUR", "amount": "0", "exponent": 2 }, "rounds": 0 }
```

4.2 Launch a demo

POST `/v2/aggregator/launch_demo` with `game_id` and optional `return_url`. No player, no token, no currency, no session: a demo never reaches your wallet. Response: `launch_url`, `game_id`. Games with `demo_supported: false` refuse with 412 (DEMO_NOT_SUPPORTED, Appendix A.2).

4.3 Catalog

POST `/v2/aggregator/games` with optional filters `query`, `vertical`, `category`, `provider`, `tags[]`, `updated_since`, `page`.

Each Game carries: `game_id`, `title`, `vertical`, `category`, `provider`, `enabled`, `demo_supported`, `wager_weight_bp`, `tags[]`, `assets[]`, `limits`, `fee_bp`, `fee_source`, `rtp`, `features[]`, `updated_at`. `wager_weight_bp` is how much a bet on this game contributes to bonus wagering and loyalty accrual, in basis points, 0 to 10000. 10000 bp = 100 %, 1000 bp = 10 %, 0 bp = 0 %. `assets[]` is {`kind`, `url`} with kinds `banner`, `thumbnail`, `background`, `icon`. `limits` is `min_bet`, `max_bet`, `max_win`. `fee_bp` is GA's fee for this game on your brand, in basis points, 0 to 10000. It's an optional field—present only once GA has resolved a fee for this game/brand pair, absent otherwise, so check presence rather than treating "absent" as 0. `fee_source` is which fee-ladder tier produced `fee_bp`: `FEE_SOURCE_UNSPECIFIED`, `FEE_SOURCE_PROVIDER`, `FEE_SOURCE_GAME`. `rtp` is the return to player (RTP) in percent, for example 96.5. The wire value is also 0 when GA has recorded no RTP for this game, so a 0 doesn't mean "0 % RTP".

`features[]` is always empty in this call. Call `get_game_features` (§6.1) to get a game's actual `features[]`.

Practical rules:

- Poll with `updated_since` (the last `updated_at` you stored) rather than reloading the whole catalog. A daily full reload is enough.
- Download `assets[].url` images to your own content delivery network (CDN) and serve them from there. Don't hotlink them from your lobby.
- Show only `enabled: true` games. A disabled game refuses to launch.

Other catalog calls are `capabilities`, `operator_capabilities`, and `update_game`. `capabilities` is what GA supports for you: currencies, settlement modes, streaming. `operator_capabilities` is what GA has on record for your wallet: `wallet_capabilities` with `rpcs`, `credit_kinds`, `debit_modes`, `notify_kinds`, `authenticate`, and `supported_transports`. `update_game` changes your own settings on a game, partial update with `update_mask` and a `reason` for audit.

4.4 Free rounds you issue

POST `/v2/aggregator/issue_grant`

Field	Required	Meaning
<code>player_ref</code> , <code>game_ids[]</code>	yes	Who and on which games.
<code>rounds_total</code>	yes	Number of free rounds.
<code>round_value</code>	yes	Stake value of one round (Money).
<code>bet_level</code> , <code>lines</code>	no	Provider bet level and lines, from <code>get_bet_ranges</code> (§6.1) where the game has levels.
<code>settlement_mode</code>	no	GA supports only <code>SETTLEMENT_MODE_ON_COMPLETION</code> . Leave it out.
<code>idempotency_key</code>	recommended	The same key returns the same grant. GA refuses a different body under the same key with <code>IDEMPOTENCY_CONFLICT</code> . Without a key every call creates a new grant.
<code>valid_until</code>	yes	Hard end date. GA refuses rounds after it.
<code>external_ref</code>	no	Your campaign or bonus id.

Response: Grant with `grant_id`, `status` (`active`, `exhausted`, `expired`, `cancelled`, `settled`), `rounds_remaining`, `rounds_played`, `accumulated_win`.

How the money flows for a grant you issued: no bet and no win reaches your wallet while the player plays the rounds. The win accumulates on the grant. When the grant is exhausted, expired, or cancelled with a win, GA sends one `settle_grant` to your wallet with `total_win`, and that's when you credit the player (§5.9). GA also settles a zero-win grant, with `total_win: 0`.

`list_grants` (by `player_ref` and `status`) and `cancel_grant` (`grant_id`, `reason`) complete the set. You can cancel only an active grant, and GA still settles the win already earned.

4.5 Sessions, rounds, and reports

Call	Send	Get back
<code>close_session</code>	<code>session_id</code> , <code>reason</code> (player request, self-exclusion, limit, operator request, suspicion, timeout)	the closed session

Call	Send	Get back
sessions	player_ref, page	sessions
rounds	session_id or player_ref, start_time, end_time, page	rounds with total_bet, total_win, status (SETTLED, REFUNDED, VOIDED, FAILED, REFUND_FAILED, see §4.6), placed_at, settled_at
export_rounds	same filters plus currency	the same rounds streamed as NDJSON lines {"result": ...}, for bulk reconciliation
aggregates	date (YYYY-MM-DD), currency	total_rounds, total_bet_amount, total_win_amount, total_refund_amount, net_ggr, total_fee

Use `aggregates` for the daily settlement check and `export_rounds` when a number doesn't match.

4.6 Event stream (optional)

`subscribe` (gRPC server stream) delivers round, grant, session, and notification events with a monotonic sequence per brand. `ack` confirms the cursor. It's an alternative to polling `rounds`, and a first integration doesn't need it.

Round events carry a `Round`. Keep the latest one by sequence per (player_ref, round_id):

event_type	Round.status	Meaning
round.settled.raw	SETTLED	The round closed without a rollback.
round.refunded	REFUNDED / VOIDED	Part or all of the round's money was rolled back.
round.failed	FAILED	Your wallet declined an operation of the round. That operation moved no money. <code>total_bet</code> / <code>total_win</code> hold the declined amount.
round.failed	REFUND_FAILED	GA owed a rollback and couldn't complete it: retries ran out, or your wallet refused the rollback. The stake is still debited. Reconcile the round manually. <code>total_bet</code> (or <code>total_win</code> for an unpaid re-issued credit) holds the stuck amount.

5. GA calls your wallet: the actions

GA sends `POST` to `https://<your-host>/v2/wallet/<action>`. Ten actions exist. Five are mandatory.

Action	Mandatory	What GA asks
<code>get_balance</code>	yes	"How much can this player play with?"
<code>debit</code>	yes	"Take the stake for this bet."
<code>credit</code>	yes	"Pay this win."
<code>rollback</code>	yes	"Undo this earlier operation."
<code>reconcile</code>	yes	"Did this operation go through?"
<code>authenticate</code>	no	"This player is opening a game. Who are they and what's their balance?"
<code>debit_credit</code>	no	"Take the stake and pay the win in one step."
<code>close_round</code>	no	"This round is over. Here are the totals."
<code>settle_grant</code>	no (mandatory if you issue free rounds)	"This free-round package is finished. Here is the total win."
<code>notify</code>	no	"Something non-monetary happened."

For an optional action you don't support, return the error envelope with `ERROR_CODE_UNSUPPORTED_ACTION` on HTTP 200 or 404. GA reads the envelope and stops calling that action. A bare 404 without the envelope is "no answer", and GA retries it.

The examples below show the envelope reduced to the fields that matter. `ts`, `operator_id`, `correlation`, and the headers are as in chapter 2. All values are illustrative.

5.1 `get_balance`

Return the playable balance in the requested currency. Change nothing.

```
{ "request_id": "req-0101", "action": "get_balance", "payload": {  
  "meta": { "request_id": "req-0101", "op_id": "op-bal-1001", "player_ref": "player-10428", "session_id":  
    "0198a1cb-4f37-7ee8-8e7d-08d39925aec0" },  
  "currency": "EUR" } }
```

```
{ "status": "ok", "data": {  
  "balance": { "currency": "EUR", "amount": 15000, "exponent": 2 },  
  "bonus_balance": { "currency": "EUR", "amount": 500, "exponent": 2 } } }
```

Refuse only for: player not found, currency mismatch, invalid request.

5.2 `authenticate` (optional)

When: the player opens a game. GA sends the `launch_token` you passed at launch. Return who the player is, the currency, and the balance. Keep the token valid: it comes back in every wallet call of the session.

```
{ "request_id": "req-0102", "action": "authenticate", "payload": {  
  "meta": { "request_id": "req-0102", "op_id": "op-auth-1002", "player_ref": "player-10428", "session_id":  
    "0198a1cb-4f37-7ee8-8e7d-08d39925aec0", "launch_token": "lt-abcdef123456" } } }
```

```
{ "status": "ok", "data": {
  "player_ref": "player-10428", "currency": "EUR",
  "balance": { "currency": "EUR", "amount": 15000, "exponent": 2 },
  "bonus_balance": { "currency": "EUR", "amount": 500, "exponent": 2 },
  "attributes": { "country": "DE", "nickname": "lucky_ann" } } }
```

Refuse for: unknown or expired token (SESSION_EXPIRED), blocked player (PLAYER_BLOCKED), player not found.

5.3 debit

Do: take the stake. Return your transaction id, the balance after, the amount debited.

Refuse when:

- amount is zero or negative: INVALID_REQUEST.
- the balance can't cover it: INSUFFICIENT_FUNDS. Final: GA doesn't resend that bet.
- the session or token is expired or missing on a call that names a session: SESSION_EXPIRED / INVALID_REQUEST.
- the player is blocked or over a limit: PLAYER_BLOCKED / LIMIT_EXCEEDED.
- the round is already closed or voided: BUSINESS_REJECTED.

Never refuse a repeated debit (Rule 1). A debit is also how a balance correction from the game provider reaches you, sometimes outside a live session. Apply it the same way.

```
{ "request_id": "req-0103", "action": "debit", "payload": {
  "meta": { "request_id": "req-0103", "op_id": "op-bet-48912", "player_ref": "player-10428", "session_id":
    "0198a1cb-4f37-7ee8-8e7d-08d39925aec0", "launch_token": "lt-abcdef123456", "round_id": "round-82721",
    "provider_ref": "spin-99182" },
  "money": { "currency": "EUR", "amount": 100, "exponent": 2 } } }
```

```
{ "status": "ok", "data": {
  "operator_tx_id": "tx-op-982103",
  "balance_after": { "currency": "EUR", "amount": 14900, "exponent": 2 },
  "debited": { "currency": "EUR", "amount": 100, "exponent": 2 } } }
```

5.4 credit

Do: add the win. Rule 3 applies: never refuse for session, block, or limits. Return your transaction id and the balance after.

- amount may be 0 for a losing spin that closes the round. Answer ok with the unchanged balance.
- kind says what the payout is: CREDIT_KIND_WIN, _BONUS, _JACKPOT, _PROMO, _FREE_ROUND, _TOURNAMENT, _CASHBACK, _ADJUSTMENT.
- components[] is an optional breakdown (KIND_BASE, KIND_JACKPOT, KIND_PROMO, KIND_BONUS, KIND_CONTRIBUTION). The parts sum to money.
- round_finish: true means this credit ends the round. Still accept a credit into a round you already closed.
- campaign_ref names the provider promo on a FREE_ROUND credit.

Refuse only for: player not found, currency mismatch, invalid request, idempotency conflict, unknown original, already rolled back.

```
{ "request_id": "req-0104", "action": "credit", "payload": {
  "meta": { "request_id": "req-0104", "op_id": "op-win-48913", "player_ref": "player-10428", "session_id":
    "0198a1cb-4f37-7ee8-8e7d-08d39925aec0", "launch_token": "lt-abcdef123456", "round_id": "round-82721",
    "provider_ref": "spin-99182" },
  "money": { "currency": "EUR", "amount": 2500, "exponent": 2 },
  "kind": "CREDIT_KIND_WIN",
  "round_finish": true } }
```

```
{ "status": "ok", "data": {
  "operator_tx_id": "tx-op-982104",
  "balance_after": { "currency": "EUR", "amount": 17400, "exponent": 2 } } }
```

5.5 debit_credit (optional)

Do: take the stake and pay the win in one atomic step. Check the stake against the current balance first, exactly like a debit. If it doesn't cover `debit_money`, refuse the whole call with `INSUFFICIENT_FUNDS` and change nothing. Never apply only one half.

```
{ "request_id": "req-0105", "action": "debit_credit", "payload": {
  "meta": { "request_id": "req-0105", "op_id": "op-step-48914", "player_ref": "player-10428", "session_id":
    "0198a1cb-4f37-7ee8-8e7d-08d39925aec0", "launch_token": "lt-abcdef123456", "round_id": "round-82721",
    "provider_ref": "spin-99183" },
  "debit_money": { "currency": "EUR", "amount": 100, "exponent": 2 },
  "credit_money": { "currency": "EUR", "amount": 300, "exponent": 2 },
  "credit_kind": "CREDIT_KIND_WIN",
  "round_finish": true } }
```

```
{ "status": "ok", "data": {
  "operator_tx_id": "tx-op-982105",
  "balance_after": { "currency": "EUR", "amount": 17600, "exponent": 2 },
  "debited": { "currency": "EUR", "amount": 100, "exponent": 2 } } }
```

5.6 rollback

Do: undo the operation named by `original_op_id`. Rule 4 applies when you don't know it. Return your transaction id, the balance after, and `original_found`.

- Rolling back a debit is always for the full stored amount: return the stake to the player.
- Rolling back a credit takes money back. `debit_mode` says what to do when the balance is too low. `DEBIT_MODE_REJECT` means refuse with `INSUFFICIENT_FUNDS`. GA's own rollbacks use this mode. `DEBIT_MODE_ALLOW_NEGATIVE` means go negative. `DEBIT_MODE_PARTIAL` means take what's there, down to zero, and report the taken amount in `debited`. `partial: true` marks a partial rollback of a credit.
- A second rollback of the same original returns the first answer and moves nothing.
- If you're still committing the original, answer an error envelope with `retryable: true`. GA comes back.
- If the player or currency doesn't match the original, refuse with `INVALID_REQUEST` or `CURRENCY_MISMATCH` and move nothing.

Never refuse a rollback for session, block, or limits. Any final refusal you do send (for example `INSUFFICIENT_FUNDS` under `DEBIT_MODE_REJECT`) stops GA's retries at once and puts the operation into manual review on GA's side.

```
{ "request_id": "req-0106", "action": "rollback", "payload": {
  "meta": { "request_id": "req-0106", "op_id": "op-rb-48915", "player_ref": "player-10428", "session_id":
    "0198a1cb-4f37-7ee8-8e7d-08d39925aec0", "launch_token": "lt-abcdef123456", "round_id": "round-82721",
    "provider_ref": "spin-99182" },
  "original_op_id": "op-bet-48912",
  "original_kind": "ORIGINAL_KIND_DEBIT",
  "money": { "currency": "EUR", "amount": 100, "exponent": 2 } } }
```

```
{ "status": "ok", "data": {
  "operator_tx_id": "tx-op-982106",
  "balance_after": { "currency": "EUR", "amount": 17700, "exponent": 2 },
  "original_found": true } }
```

Unknown original: same shape with `"original_found": false`, no money moved, and the id stored as rolled back.

5.7 `close_round` (optional)

Do: record that the round is over. `net_win`, `bet_total`, `win_total` are for your reports. No money moves on this call.

```
{ "request_id": "req-0107", "action": "close_round", "payload": {
  "meta": { "request_id": "req-0107", "op_id": "op-close-48916", "player_ref": "player-10428", "round_id":
    "round-82721" },
  "net_win": { "currency": "EUR", "amount": 200, "exponent": 2 },
  "bet_total": { "currency": "EUR", "amount": 100, "exponent": 2 },
  "win_total": { "currency": "EUR", "amount": 300, "exponent": 2 } } }
```

```
{ "status": "ok", "data": {
  "operator_tx_id": "tx-op-982107",
  "balance_after": { "currency": "EUR", "amount": 17700, "exponent": 2 } } }
```

5.8 `reconcile`

When: GA got no clear answer to an operation and asks what happened. Do: look up `original_op_id` and report. Change nothing and remember nothing. This is a read. Never return `UNKNOWN_ORIGINAL` here.

Your record of <code>original_op_id</code>	Return state
not there	<code>RECONCILE_STATE_NOT_APPLIED</code>
still being committed	<code>RECONCILE_STATE_UNKNOWN</code>
applied	<code>RECONCILE_STATE_APPLIED</code> with the original <code>operator_tx_id</code> and <code>balance_after</code>
rolled back, or remembered as rolled back (Rule 4)	<code>RECONCILE_STATE_NOT_APPLIED</code>

```
{ "request_id": "req-0110", "action": "reconcile", "payload": {
  "meta": { "request_id": "req-0110", "op_id": "op-rec-48918", "player_ref": "player-10428" },
  "original_op_id": "op-bet-48912" } }
```

```
{ "status": "ok", "data": {
  "state": "RECONCILE_STATE_APPLIED",
  "operator_tx_id": "tx-op-982103",
  "balance_after": { "currency": "EUR", "amount": 14900, "exponent": 2 } } }
```

5.9 `settle_grant` (mandatory if you issue free rounds)

When: a grant you issued with `issue_grant` (§4.4) is finished: exhausted, expired, or cancelled with a win. Do: credit `total_win` to the player, once per `grant_id`. The call is idempotent by `op_id`. GA sent no debits and no credits for the rounds, so this call is the only place the win reaches the player. `total_win: 0` needs only an `ok`. Exception (Money Path Rules §2.10 item 7): a win of a round played before the grant ended can reach GA after the grant ended (settled via `settle_grant`, or cancelled or expired before payout). Such a win arrives as a separate credit with `kind = CREDIT_KIND_FREE_ROUND` and `campaign_ref` set to the `grant_id`, idempotent by `op_id` like any other credit. A new round reported on a grant that has already ended is still refused to GA and never reaches you.

Two free-round mechanisms exist and they never overlap for the same win:

Mechanism	Who starts it	How the win reaches your wallet
Grant you issue (<code>issue_grant</code>)	you	one <code>settle_grant</code> with <code>total_win</code> at the end
Promo free spins from the game provider	the provider or a GA campaign	an ordinary credit with <code>CREDIT_KIND_FREE_ROUND</code> and <code>campaign_ref</code> per winning spin

Free-round wins may arrive with no `session_id` and no `launch_token`. Accept them (Rule 3).

```
{ "request_id": "req-0108", "action": "settle_grant", "payload": {  
  "meta": { "request_id": "req-0108", "op_id": "op-grant-7781", "player_ref": "player-10428" },  
  "grant_id": "grant-7781",  
  "total_win": { "currency": "EUR", "amount": 1250, "exponent": 2 },  
  "rounds_played": 20 } }
```

```
{ "status": "ok", "data": {  
  "operator_tx_id": "tx-op-982108",  
  "balance_after": { "currency": "EUR", "amount": 18950, "exponent": 2 } } }
```

5.10 notify (optional)

Do: record the event and answer `ok`. Nothing to move. GA sends it once and doesn't retry.

Kinds: `NOTIFY_KIND_FREE_ROUNDS_STARTED`, `_TOKEN_RESET`, `_SESSION_EXPIRED`, `_REALITY_CHECK`, `_PRIZE_AWARDED`, `_FREESPIN_ISSUED`, `_FREESPIN_FINISHED`, `_BONUS_BUY_FINISHED`.

```
{ "request_id": "req-0109", "action": "notify", "payload": {  
  "meta": { "request_id": "req-0109", "op_id": "op-ntf-48917", "player_ref": "player-10428", "session_id":  
    "0198a1cb-4f37-7ee8-8e7d-08d39925aec0" },  
  "kind": "NOTIFY_KIND_SESSION_EXPIRED",  
  "data": { "attributes": { "reason": "idle_timeout" } } } }
```

```
{ "status": "ok", "data": { "operator_tx_id": "ntf-op-1" } }
```

6. Optional: provider features

POST `https://api.rexplay.site/v2/features/<call>`, signed as in §2.3. Each game supports only some features. Call `get_game_features` first and look at `features[]`. Postman: `postman/GA_Features_API_v2.postman_collection.json`.

6.1 The calls

What you want	Call	Send	Get back
Which features does this game support?	<code>get_game_features</code>	<code>game_code</code> , <code>provider_slug</code>	<code>features[]</code> (<code>FEATURE_FREE_ROUNDS_GRANT</code> , <code>FEATURE_BONUS_BUY</code> , <code>FEATURE_JACKPOT_FEED</code> , <code>FEATURE_BET_RANGES</code> , <code>FEATURE_ROUND_REPLAY</code> , <code>FEATURE_TOURNAMENT</code> , <code>FEATURE_DEMO</code> , ...), <code>limits</code>
Allowed stake levels for a game	<code>get_bet_ranges</code>	<code>provider_slug</code> , <code>game_code</code> , <code>currency</code>	<code>bet_ranges[]</code> with <code>bet_level</code> , <code>bet_value</code> , <code>min_bet</code> , <code>max_bet</code> , <code>step</code> . Use <code>bet_level</code> in <code>issue_grant</code> .
Bonus buy	<code>list_bonus_buy_types</code> , <code>issue_bonus_buy</code> , <code>cancel_bonus_buy</code>	<code>provider_slug</code> , <code>game_code</code> , <code>player_ref</code> , <code>bonus_type</code> , <code>amount</code> , <code>currency</code> , <code>attributes["idempotency_key"]</code>	<code>provider_ref</code>
Current jackpot values	<code>get_jackpots</code>	<code>provider_slug</code> , <code>currency</code>	<code>jackpots[]</code>
Replay or details of a disputed round	<code>get_round_replay</code> , <code>get_round_details</code>	<code>provider_slug</code> , <code>round_id</code> , <code>player_ref</code>	<code>replay_url</code> / <code>details</code>
Provider campaigns	<code>list_campaigns</code> , <code>cancel_campaign</code>	<code>provider_slug</code> , <code>campaign_id</code>	<code>campaigns</code>
Tournaments	<code>list_tournaments</code> , <code>get_tournament</code> , <code>get_leaderboard</code>	<code>provider_slug</code> , <code>tournament_id</code>	<code>tournaments</code> , <code>entries[]</code>
Provider-side statement for reconciliation	<code>query_provider_transactions</code>	<code>provider_slug</code> , <code>player_ref</code> , <code>date_from</code> , <code>date_to</code>	<code>transactions[]</code> , <code>totals</code>

Errors: the `{"code", "message", "retryable"}` body of §2.8 with a non-200 status, Appendix A.2.

7. Data formats

Value	Format
Currency	Uppercase letters and digits, 3 to 10 characters. International Organization for Standardization (ISO) 4217 for fiat (EUR), exchange ticker for crypto (USDT). You and GA agree the exponent per currency at onboarding.
Money	{currency, amount, exponent}. See §2.6. Number on your wallet, string on GA's APIs.
Timestamps	RFC 3339 in Coordinated Universal Time (UTC): 2026-09-13T12:00:00Z. aggregates.date is YYYY-MM-DD.
Language	Two lowercase letters: en, de. Regional forms (en-GB) fall back to en.
Ids (player_ref, op_id, round_id, game_id, session_id)	Opaque strings. GA sets no length limit. Keep yours under 128 characters. GA's session ids are universally unique identifiers (UUIDs).
wager_weight_bp, fee_bp	Basis points, 0 to 10000.

Unknown fields in any message must be ignored. Unknown enum values in a known field are refused with `INVALID_REQUEST`.

8. Network and limits

- Your wallet must answer within 5 seconds per call (§2.7). Aim for well under 1 second.
 - GA doesn't filter your source IP on the Operator API. If you restrict inbound traffic to your wallet, allow GA's outbound addresses: `49.13.169.177` and `46.224.156.1` for the sandbox, `49.13.169.177` for production.
 - GA enforces no request rate limit on the Operator API today. Keep catalog polling to once a minute or less and use `updated_since`. GA reserves a 429 with `ERROR_CODE_RATE_LIMITED` for abuse.
 - Secrets never go into source control, browser code, tickets, or logs.
-

9. Certification and go-live

9.1 Run the checks from the Operator Portal

Log in to the Operator Portal with an `operator_admin` account and open Integration. The Callback test section runs the certification checks against your sandbox wallet:

1. Save your wallet endpoint URL. For the v2 contract the URL must contain `/v2` (for example `https://wallet.operator.example/v2/wallet`). That's how the portal picks the v2 suite.
2. Create a test player with the `person_add` button. The portal creates and funds it. The player reference doubles as the launch token in the run.
3. Choose Full suite, set the currency and bet amount, and press Run suite.

The run shows every check with its pass/fail state, balance deltas, and a hint for each failure. You can replay bet and refund steps one by one. The checks are the same 55 `W2-*` checks the command-line tool runs. Six of them (`W2-D1` to `W2-D6`) test the four rules of chapter 3. You're done when every check passes.

9.1a The command-line tool (for your CI)

The same suite exists as `ga-operator-certify` for automated runs in your pipeline. Ask your GA integration manager for the binary for your platform.

```
./ga-operator-certify \
--binding http \
--target https://wallet.operator.example \
--api-key key-cert-01 \
--secret sec-cert-01 \
--strict
```

Before running, create in your wallet a test player `certify-player-001` (or pass `--player-ref`) with a funded EUR balance (or `--currency`), and a valid launch token for that player passed with `--launch-token` (default `token-cert-001`). The tool creates neither. `--json` writes a report with the request and response of every failed check.

9.2 The checklist

Go through `certification/GA_Operator_Certification_Checklist_v2.md` with your GA integration manager and sign it on both sides. Then repeat the Postman run, the tool, and the checklist against your production wallet with production credentials.

9.3 Changelog

Contract changes are additive only, and GA lists them in `CHANGELOG.md` in this package. GA never removes or renames anything published. New optional fields may appear at any time, which is why your parser must ignore unknown fields.

Appendix A. Every error code

A.1 Codes your wallet returns to GA

Always HTTP 200 with `{"status": "error", "error": {"code": ...}}` (§2.5). 15 codes. The list is closed.

Code	GA retries?	When to return it
ERROR_CODE_INSUFFICIENT_FUNDS	no	The balance can't cover the stake, or a credit rollback under DEBIT_MODE_REJECT
ERROR_CODE_PLAYER_NOT_FOUND	no	No such player
ERROR_CODE_PLAYER_BLOCKED	no	Player blocked, self-excluded, or disabled. New bets only. Still accept wins and rollbacks
ERROR_CODE_CURRENCY_MISMATCH	no	Currency or exponent doesn't match the player's account
ERROR_CODE_BUSINESS_REJECTED	no	Your own business rule refused it. Also a new bet into a closed or voided round
ERROR_CODE_UNKNOWN_ORIGINAL	no	A credit names an original you don't have. Never for a rollback or a reconcile
ERROR_CODE_ALREADY_ROLLED_BACK	no	You already rolled back the <code>op_id</code> (Rule 4), or the round is voided
ERROR_CODE_UNSUPPORTED_ACTION	no	You don't implement this optional action (HTTP 200 or 404, with the envelope)
ERROR_CODE_RATE_LIMITED	yes	You are rate-limiting GA (HTTP 200 or 429, with the envelope)
ERROR_CODE_MAINTENANCE	yes	Your wallet is down for maintenance
ERROR_CODE_INVALID_REQUEST	no	Malformed request: missing field, zero or negative amount, overflow, unknown enum value, missing token on a session call
ERROR_CODE_SESSION_EXPIRED	no	Launch token or session expired. New bets only
ERROR_CODE_LIMIT_EXCEEDED	no	Deposit or loss limit reached. New bets only
ERROR_CODE_INTERNAL	yes	Unexpected error on your side (HTTP 200 or 500, with the envelope)
ERROR_CODE_IDEMPOTENCY_CONFLICT	no	Same <code>op_id</code> , different content (Rule 1)

Bare statuses without the envelope (401, 403, 404, 429, 5xx) are all “no answer”: GA retries, and a bet ends in a rollback.

A.2 What GA returns to you

Every `/v2/aggregator/*` and `/v2/features/*` error is the body of §2.8. `message` starts with the token in the table.

HTTP	code	Retry?	message starts with	Meaning
400	ERROR_CODE_INVALID_REQUEST	no	invalid json body, invalid request, invalid_request, invalid_provider_ref, capability_not_supported, bonus_buy_not_cancellable	Malformed JSON, missing or invalid field, GA can't read the body, or the game provider doesn't offer the capability
401	ERROR_CODE_INVALID_REQUEST	no	unauthorized, signature_missing, signature_invalid, signature_stale	Wrong or missing key id, signature, or a timestamp outside 300 s
403	ERROR_CODE_BUSINESS_REJECTED	no	forbidden	The operator is suspended, or the key isn't allowed to do this (for example a read-only key on launch_game)
403	ERROR_CODE_BUSINESS_REJECTED or ERROR_CODE_INVALID_REQUEST	no	operator_suspended, operator_forbidden, restricted, game_unavailable, PLAYER_SELF_EXCLUDED, PLAYER_COOL_OFF	Not allowed for this operator, game, or player
404	ERROR_CODE_INVALID_REQUEST	no	provider not found, provider_not_found, game not found, player_not_found, bonus_buy_not_found	Provider, game, player, grant, or bonus not found
409	ERROR_CODE_IDEMPOTENCY_CONFLICT	no	session_token_conflict	You reused the token of launch_game with another player, game, or currency, or after its session closed or expired, or the idempotency_key of issue_grant with a different body (§4.1, §4.4)
412	ERROR_CODE_INVALID_REQUEST	no	DEMO_NOT_SUPPORTED	launch_demo on a game with demo_supported: false
422	ERROR_CODE_INVALID_REQUEST	no	unsupported action	Action not supported for this game or provider
429	ERROR_CODE_RATE_LIMITED	yes		Rate limit
500	ERROR_CODE_INTERNAL	no	internal error	GA-side failure. GA gets an alert. Contact support if it repeats
501	ERROR_CODE_UNSUPPORTED_ACTION	no		Call not implemented on this binding (settlement_mode PER_ROUND)
502	ERROR_CODE_INTERNAL		provider_error	

HTTP	code	Retry?	message starts with	Meaning
		no (retryable not set)		The game provider returned an error. Repeating the same call returns the same error
503	ERROR_CODE_MAINTENANCE	yes	provider_unavailable	The game provider is unavailable
503	ERROR_CODE_INTERNAL	yes	launch_in_progress	The first launch_game with this token is still running. Retry with the same token (§4.1)
504	ERROR_CODE_INTERNAL	yes	provider_timeout	The game provider timed out

Retry only 429, 503, and 504, with backoff.

Appendix B. Numbers to remember

What	Value
GA waits for one wallet call	5 seconds
Quick attempts on “no answer”	up to 4, spaced 100 ms, 500 ms, 2 s, inside an 8 s budget per bet (a silent wallet gets 2)
Background resend of a credit, rollback, or grant settlement	up to 72 hours, backoff 1 s doubling to a 10-minute cap
Retry-After honoured up to	10 seconds inside the quick attempts
Signature clock window, both directions	±300 seconds
GA outbound IPs	49.13.169.177, 46.224.156.1 (sandbox). 49.13.169.177 (production)
Keep op_id records for	4 months
Game session lifetime	24 hours
Money exponent	0 to 18, from the message. GA sends 2 or more
Catalog page size	50 by default, 500 maximum
Wallet HMAC secret shown	once, at rotation in the Operator Portal
Certification checks	55, of which 6 are money-path checks