YM Trading API v2 Pilot

Participant-facing REST, private execution feed, market identity, recovery, error and capacity contract for approved integrations.

Status	Controlled pilot
Reviewed	2026-09-02
REST base	https://ympredictions.com
Private feed	wss://ympredictions.com/v1/trading-feed
OpenAPI	openapi-market-maker.yaml

This handbook documents implemented pilot behavior and labels preview or in-development capabilities explicitly. It is not a service-level agreement.

Contents

Overview

Status: **controlled pilot** Last reviewed: **2026-09-02**

This package describes the external interface currently available to approved YM market-making and API-trading participants. It is participant-facing documentation, not an architecture plan and not a service-level agreement.

Contract activation is environment-specific. Do not submit v2 commands until YM confirms that v2 is active for the supplied endpoint and account.

YM uses DXmatch as the matching and venue-order authority. Clients authenticate to YM and use YM APIs; direct DXmatch credentials are not part of the pilot. YM remains responsible for participant identity, admission, collateral, idempotency, market-generation fencing, account projections, ledger posting, market lifecycle and settlement.

Availability language

Label	Meaning
Available	Implemented and enabled for approved accounts.
Pilot	Implemented and available to explicitly onboarded participants, but not yet covered by a public production SLA.
Preview	Useful for evaluation, but its recovery or capacity contract is not suitable for production market making.
In development	Planned or being built; clients must not depend on it.
Not offered	Deliberately outside the current external contract.

Current pilot surface

Capability	Status
Canonical event, family and market discovery	Available
Public L2 book snapshots, recent trades and market statistics	Available
OAuth 2.0 client-credentials authentication	Pilot onboarding
Canonical REST `NEW`, `REPLACE` and `CANCEL`	Pilot
Batch `NEW` and batch `CANCEL`	In development; excluded from the v2 pilot until per-item replay conformance passes
Canonical order, position, portfolio and balance reads	Pilot
Participant trade-history read	Pilot; legacy YES-book coordinate, not private-feed recovery authority
Resumable private order and execution WebSocket feed	Pilot
Consumer market-data WebSocket	Preview; not a professional recovery contract
Persistent WebSocket order entry	In development
Professional recoverable streaming market data	In development
FIX order entry or drop copy	In development; not part of this pilot
Mass cancel, cancel-on-disconnect and participant kill switch	In development
External mass-quote or quote-set replacement	Not offered in this pilot

Documents

1. Quickstart
2. OpenAPI guide and curated OpenAPI
3. Authentication and scopes
4. Markets, identity and lifecycle
5. Order entry and lifecycle
6. Private trading feed
7. Errors and retry policy
8. Market data
9. Limits, availability and roadmap

Contract boundaries

- Order-command, canonical-order, and private-feed fill prices are integer cents from `0` through `100` in the named outcome's coordinate. Quantity is an integer number of contracts.
- Orders use `action` (`BUY` or `SELL`) plus `outcome` (`YES` or `NO`). YM derives venue side and position effect; clients do not send either.
- An HTTP admission response is not proof that an order is working at the venue. Exchange-effective state is reported by the private trading feed and canonical order reads.
- Clients must use immutable `market_id` values and the current positive `market_generation` returned by the canonical catalog.
- Participant identity comes only from the validated bearer token. A client cannot select another participant through a request body, query parameter or WebSocket cursor.
- Delivery on the private feed is at least once. Consumers deduplicate by `event_id` and converge each order using `order_version`.
- Current measured latencies and configured limits are pilot evidence, not a contractual SLO. Contractual capacity will be published after sustained conformance and load testing.

v2 transition

The public URL namespace remains `/v1`; `schema_version: 2` versions the order and private-feed payload contract. v2 replaces the former three-axis order input with one consumer-facing model: `action`, `outcome`, and the selected outcome's `price_cents`. The retired public fields are `side`, `position_side`, `intent`, and decimal `price`. They are not compatibility aliases and v2 rejects them.

REST order resources and private-feed fills use the same selected-outcome price coordinate. Private-feed messages carry `schema_version: 2`; a v1 cursor cannot resume v2. Before cutover, consumers must durably finish their v1 stream through the announced boundary. A cursorless v2 connection then replaces nonterminal order state only; it does not replay v1 terminal orders or fill history. Internal matching, collateral, ledger, and settlement semantics are unchanged. The public L2 book and public trade tape remain YES-book market data. The current authenticated `/v1/trades` history also exposes venue side and YES-book price; use the v2 private feed for live participant-relative fill truth.

Forward path

The REST contract and private feed are suitable for a controlled integration now. Before general professional availability, YM intends to add persistent WebSocket command entry, a professional recoverable market-data transport, cancel-safety primitives, generalized firm/strategy onboarding, high- availability proof and published capacity/SLO terms. New transports will adapt the same canonical command and event model rather than create separate trading logic.

Sources of truth

For the pilot, use this order when documents appear to disagree:

1. The current curated OpenAPI in this directory.
2. Runtime behavior confirmed in the participant UAT environment.
3. This explanatory documentation.
4. Roadmap notes marked **In development**.

Report contract discrepancies to the YM integration contact supplied during onboarding. Include the response status, machine-readable error code, UTC time, request correlation ID and client request ID. Never send client secrets or bearer tokens.

---

1. Quickstart

Status: **controlled pilot**

This guide uses the v2 order contract. Confirm v2 activation for the target environment before sending commands; v1 fields are intentionally rejected.

This sequence opens the private feed before submitting an order, then follows one order through `NEW`, `REPLACE` and `CANCEL`. Use an account and market explicitly approved for UAT. Do not run the example against an unapproved financial environment.

Prerequisites

YM supplies each approved integration with:

- an OAuth confidential `client_id` and `client_secret`;
- the YM account ID bound to that client;
- the granted scopes;
- the permitted environment and test markets; and
- an operational contact.

The initial pilot binds one confidential client to one financial account. Do not place a `user_id` in an order request.

Set local variables without committing their values:

```
sh
export YM_BASE_URL="https://ympredictions.com"
export YM_TOKEN_URL="https://auth.ympredictions.com/realms/ymp/protocol/openid-connect/
token"
export YM_CLIENT_ID="supplied-by-ym"
export YM_CLIENT_SECRET="supplied-by-ym"
export YM_ACCOUNT_ID="supplied-by-ym"
```

1. Obtain a short-lived access token

```
sh
TOKEN_RESPONSE="$(curl -sS -X POST "$YM_TOKEN_URL" \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  --data-urlencode 'grant_type=client_credentials' \
  --data-urlencode "client_id=$YM_CLIENT_ID" \
  --data-urlencode "client_secret=$YM_CLIENT_SECRET")"

export YM_ACCESS_TOKEN="$(printf '%s' "$TOKEN_RESPONSE" | jq -r '.access_token')"
```

The current pilot token lifetime is five minutes. Request a new token before expiry; do not expect a refresh token from the client-credentials grant.

2. Select a canonical market

```
sh
curl -sS \
  "$YM_BASE_URL/v1/markets?view=canonical&status=active&limit=20" \
  | jq '.markets[] | {
      market_id: .id,
      market_generation,
      status,
      publication_state,
      title
    }'
```

Select a leaf with all of the following:

- a positive `market_generation`;
- `status` equal to `OPEN`;
- `publication_state` equal to `PUBLIC_OPEN`; and
- a close time that has not passed.

The order API remains the final admission authority. A catalog result is not a reservation of future tradeability.

```
sh
export YM_MARKET_ID="selected-immutable-market-id"
export YM_MARKET_GENERATION="selected-positive-generation"
```

3. Read the current L2 snapshot

```
sh
curl -sS \
  "$YM_BASE_URL/v1/book?market_id=$YM_MARKET_ID&depth=20" \
  | jq .
```

The snapshot is public. `seq` describes the current YM projection; it is not a durable professional replay cursor. See Market data.

4. Open the private feed

Open this in a second terminal before sending the order:

```
sh
npx wscat \
  -c 'wss://ympredictions.com/v1/trading-feed' \
  -H "Authorization: Bearer $YM_ACCESS_TOKEN"
```

Wait for this bootstrap sequence:

```
text
stream.welcome
stream.snapshot_started
stream.snapshot_chunk (zero or more)
stream.snapshot_completed
```

```
stream.ready
```

Only submit commands after `stream.ready`. Persist each event's signed `cursor` after the event has been validated and durably applied locally.

5. Submit a resting order

The example buys ten YES contracts at 45 cents using `GTC`:

```
sh
export YM_CLIENT_ORDER_ID="quickstart-order-$(date +%s)"
export YM_NEW_REQUEST_ID="quickstart-new-$(date +%s)-$RANDOM"

NEW_RESPONSE="$(curl -sS -X POST "$YM_BASE_URL/v1/orders" \
  -H "Authorization: Bearer $YM_ACCESS_TOKEN" \
  -H 'Content-Type: application/json' \
  -H "Idempotency-Key: $YM_NEW_REQUEST_ID" \
  --data "{
    \"client_order_id\": \"$YM_CLIENT_ORDER_ID\",
    \"market_id\": \"$YM_MARKET_ID\",
    \"market_generation\": $YM_MARKET_GENERATION,
    \"action\": \"BUY\",
    \"outcome\": \"YES\",
    \"price_cents\": 45,
    \"quantity\": 10,
    \"time_in_force\": \"GTC\"
  }")"

printf '%s' "$NEW_RESPONSE" | jq .
export YM_ORDER_ID="$(printf '%s' "$NEW_RESPONSE" | jq -r '.order.id')"
```

`201 Created` means YM admitted the command and durably created a `PENDING` order. Wait for `order.working` on the private feed, or read the canonical order until `execution_status` is `WORKING`, before treating it as venue-effective.

For a NO order, keep the same coordinate end to end. A request with `"action":"BUY"`, `"outcome":"NO"`, and `"price_cents":35` must return and stream `BUY` / `NO` / `35`. Never send or expect the internal complementary YES-book representation.

6. Read the canonical order

```
sh
curl -sS \
  "$YM_BASE_URL/v1/orders/$YM_ORDER_ID" \
  -H "Authorization: Bearer $YM_ACCESS_TOKEN" \
  | jq .
```

Record `order_version`. A replacement is fenced by the version it was based on.

7. Replace the working order

This changes the effective price of the order's immutable YES outcome to 44 cents while retaining the target total quantity of ten contracts:

```
sh
ORDER_RESPONSE="$(curl -sS \
  "$YM_BASE_URL/v1/orders/$YM_ORDER_ID" \
  -H "Authorization: Bearer $YM_ACCESS_TOKEN")"
export YM_ORDER_VERSION="$(printf '%s' "$ORDER_RESPONSE" | jq -r '.order_version')"
export YM_REPLACE_REQUEST_ID="quickstart-replace-$(date +%s)-$RANDOM"

curl -sS -X POST \
  "$YM_BASE_URL/v1/orders/$YM_ORDER_ID/replace" \
  -H "Authorization: Bearer $YM_ACCESS_TOKEN" \
  -H 'Content-Type: application/json' \
  -H "Idempotency-Key: $YM_REPLACE_REQUEST_ID" \
  --data "{
```

```
    \"expected_order_version\": $YM_ORDER_VERSION,
    \"price_cents\": 44,
    \"total_quantity\": 10
  }" | jq .
```

`202 Accepted` means the replacement command was admitted. The prior venue-effective price and quantity remain authoritative until `order.replaced` arrives.

8. Cancel the order

Do not send a cancel while a replacement is still pending. Wait for `order.replaced`, then use a new idempotency key:

```
sh
export YM_CANCEL_REQUEST_ID="quickstart-cancel-$(date +%s)-$RANDOM"

curl -sS -X POST \
  "$YM_BASE_URL/v1/orders/$YM_ORDER_ID/cancel" \
  -H "Authorization: Bearer $YM_ACCESS_TOKEN" \
  -H 'Content-Type: application/json' \
  -H "Idempotency-Key: $YM_CANCEL_REQUEST_ID" \
  --data '{}' | jq .
```

`202 Accepted` is not cancellation finality. Wait for `order.cancelled` or a canonical terminal snapshot before releasing local risk.

9. Resume after disconnect

Reconnect with the last cursor your application fully applied:

```
text
wss://ympredictions.com/v1/trading-feed?cursor=<URL-encoded-signed-cursor>
```

The server replays events after that cursor and sends `stream.ready`. Delivery is at least once; deduplicate using `event_id` and never derive sequence numbers from the cursor.

Forward path

REST is the pilot command transport. Persistent WebSocket order entry is in development and will use the same IDs, idempotency, market-generation fence and canonical lifecycle. Existing REST clients will not need a separate business model when that transport becomes available.

---

2. Curated OpenAPI

Status: **available for pilot integration**

The standalone openapi-market-maker.yaml contains only the endpoints intended for approved API-trading participants. It is derived from YM's authoritative platform OpenAPI rather than being a second handwritten contract.

Included surface

The curated specification includes:

- canonical taxonomy, event, family and market discovery;
- market detail, L2 book snapshots, recent trades and statistics;

- canonical order creation, listing, lookup, replacement and cancellation;
- the private order and execution WebSocket handshake and message schemas;
- positions, portfolio and self-scoped balances; and
- the bearer-token security scheme and recursively required schemas.

Deliberately excluded

The participant specification does not expose:

- admin and operator routes;
- payments, withdrawals or responsible-gambling administration;
- internal service callbacks;
- internal Paper-LP quote-set replacement;
- batch commands pending per-item replay and error-envelope conformance;
- legacy authenticated `/v1/trades`, whose YES-book activity projection is not Order Contract v2 or private-feed recovery authority;
- user-created index management;
- provider anchor administration; or
- planned WebSocket/FIX command endpoints that do not exist yet.

The absence of a path from this specification means it is not part of the market-maker pilot, even if another YM application uses that path internally.

Tooling

The file is OpenAPI 3.1 and can be imported into current OpenAPI tooling. For example:

```
sh
npx @redocly/cli lint openapi-market-maker.yaml
```

Client generation is optional. YM recommends first integrating directly with the JSON contract so retry, idempotency and asynchronous effective-state semantics remain explicit.

Versioning

- The current public order and private-feed schema is v2.
- Additive optional response fields may be introduced within v2.
- Existing field meanings, enum meanings, precision, identity or recovery behavior will not be silently changed.
- A breaking semantic change requires a new major contract version and a migration notice.
- Unknown response fields should be ignored. Unknown command enum values must not be sent.

The v2 cutover is intentionally breaking: old order fields and v1 private-feed cursors are not translated. A cursorless v2 snapshot replaces nonterminal state only; retain v1 terminal/fill history through the announced boundary. Confirm environment activation before sending v2 traffic.

Forward path

The current artifact is checked into the repository as a curated snapshot of the authoritative specification. Before a public developer portal is launched, YM intends to enforce allowlist generation and drift detection in CI, publish a rendered reference site, and attach an explicit changelog to each external contract release. Runtime route-contract tests already protect the underlying platform specification.

3. Authentication and scopes

Status: **controlled pilot onboarding**

Professional clients authenticate to YM with OAuth 2.0 client credentials. The bearer token identifies both the approved confidential client and, during the initial pilot, one dedicated YM financial account.

Direct DXmatch credentials, browser passwords, API keys in URLs and shared credentials across firms are not supported.

Token endpoint

```
text
POST https://auth.ympredictions.com/realms/ymp/protocol/openid-connect/token
Content-Type: application/x-www-form-urlencoded
```

```
sh
curl -sS -X POST \
  'https://auth.ympredictions.com/realms/ymp/protocol/openid-connect/token' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  --data-urlencode 'grant_type=client_credentials' \
  --data-urlencode 'client_id=<supplied-client-id>' \
  --data-urlencode 'client_secret=<supplied-client-secret>'
```

Use the returned token on REST requests:

```
text
Authorization: Bearer <access_token>
```

The same header authenticates a headless connection to `wss://ympredictions.com/v1/trading-feed`.

Pilot scopes

Scope	Authority
`orders:write`	Submit approved `NEW`, `REPLACE` and `CANCEL` commands. Batch commands remain outside the v2 pilot.
`orders:read`	Read the authenticated participant's orders, trades, positions and portfolio.
`trading-feed:read`	Connect to the authenticated participant's private lifecycle feed.
`balances:read`	Read balances for the account bound to the token.
`trading:professional`	Select the explicitly configured professional capacity policy; it grants no data or trading authority by itself.

Scopes are assigned by YM during onboarding. A client cannot elevate itself by placing additional values in the token request.

Identity and account binding

- Do not send `user_id` in an order body.
- Optional `user_id` filters on authenticated read routes may only identify the token's own account; omitting the filter is preferred.

- `/v1/accounts/{userID}/balances` returns data only when `{userID}` is the token-bound account, unless the caller is a separate authorized YM operator.
- Private-feed identity is derived from the bearer token. There is no account selector in the WebSocket query string.
- A signed replay cursor is bound cryptographically to the participant that received it and cannot be used by another participant.

Token lifecycle

The current pilot access-token lifetime is five minutes. Treat that as an operational setting, not a permanent protocol guarantee.

- Cache the token only in process memory or an approved secret store.
- Request a replacement token before expiry.
- Establish a new WebSocket connection with the replacement token and resume from the last durably applied cursor.
- Never log tokens, client secrets or full authorization headers.
- Do not assume a client-credentials refresh token is available.

Immediate credential revocation and rotation are handled through the pilot onboarding contact. A revoked token must not be retried indefinitely.

Browser and headless clients

The private feed accepts authenticated headless clients without a browser `Origin` header. Browser Origin protections remain active where applicable and are not weakened by professional access. The current consumer market-data WebSocket has a different browser-oriented policy and is not the professional market-data contract.

Forward path

General professional availability will add explicit firm, application, strategy and account entitlements; self-service rotation and revocation; connection/subscription policy; and an audited onboarding workflow. mTLS may be added if operational or regulatory requirements justify it, but is not a pilot requirement. Future WebSocket or FIX command sessions will resolve to the same YM principal and account authority.

---

4. Markets, identity and lifecycle

Status: canonical discovery **available**; dedicated professional lifecycle stream **in development**

Every trading command targets one immutable leaf `market_id` and the exact current `market_generation`. Titles, slugs, URLs and provider identifiers are presentation or discovery data and must not be used as trading identity.

Hierarchy

Resource	Purpose
Event	The real-world occurrence, fixture, election or question grouping.
Market family	A coordinated proposition such as match winner, set winner or tournament winner.
Market leaf	One binary outcome market. Public orders select YES or NO; YM normalizes both to the leaf's internal YES book.

A family may contain one or many leaves. `membership_complete` describes whether the canonical family membership is complete; it does not make the entire family one tradable instrument.

Canonical discovery

Always request the canonical view:

```
text
GET /v1/markets?view=canonical
GET /v1/markets/{marketID}?view=canonical
GET /v1/events
GET /v1/events/{eventID}
GET /v1/events/{eventID}/market-families
GET /v1/market-families?view=canonical
GET /v1/market-families/{familyID}?view=canonical
```

Canonical collection pagination uses the opaque `next_cursor`; offset pagination is unsupported. Treat cursors as opaque and scoped to the request shape that produced them.

Provider mappings are returned only through explicit canonical expansions. They are diagnostic metadata, not stable order-entry identifiers.

Trading eligibility indicators

Before submitting an order, require:

- `market_generation > 0`;
- market `status` equal to `OPEN`;
- `publication_state` equal to `PUBLIC_OPEN`; and
- `close_at`, when present, later than the current UTC time.

These are client-side preflight indicators. YM rechecks authoritative admission, lifecycle, participant and collateral state atomically during order entry. A market can stop accepting commands after it was discovered.

`PUBLIC_BLOCKED` may remain visible for discovery, live presentation or history while admission is disabled. It is not a tradable state.

Market generation

`market_generation` is an optimistic safety fence around the market's current admission definition. It prevents a command constructed against an older definition from entering a newer market state.

On `409 stale_market_generation`:

1. Stop sending commands with the old generation.
2. Fetch `GET /v1/markets/{marketID}?view=canonical`.
3. Re-evaluate status, publication, close time and the changed definition.
4. Rebuild the command with the new positive generation.
5. Use a new client request ID if the economic command changed. Reuse the old ID only to recover the exact original payload.

Never increment or infer a generation locally.

Lifecycle behavior

- An `OPEN` market may accept orders when its admission and publication state also allow trading.
- Closing begins a controlled stop-and-drain process. A displayed live status alone does not override admission state.

- `CLOSED` prevents new exposure while finality or settlement may still be in progress.
- `RESOLVED` records the canonical result. Historical discovery may retain the market after trading has ended.
- Existing nonterminal orders may receive system-generated cancellation or expiry events as part of lifecycle control. These appear on the same private feed as client commands.

Do not assume the scheduled start of an event is necessarily its trading close or resolution time. Use the canonical market fields and effective lifecycle events.

Revision versus generation

`revision` tracks canonical presentation/catalog changes. `market_generation` is the trading admission fence. A client must not substitute one for the other.

Forward path

Catalog REST is authoritative for the pilot. YM intends to add a professional market-lifecycle stream carrying market identity, generation, admission and close/drain transitions so clients do not need frequent catalog polling. That stream will be wake-driven and recoverable; it will not create a second source of market truth.

---

5. Order entry and lifecycle

Status: canonical REST `NEW`, `REPLACE` and `CANCEL` **available in the controlled pilot**

YM exposes one canonical order model. REST admission, canonical reads and the private feed describe the same order and command identities. Future WebSocket or FIX adapters will enter this same business path rather than create separate order semantics.

Three distinct states

Never collapse these layers into one generic `accepted` state:

Layer	Authority	Meaning
Command admission	YM	The command was durably accepted or rejected by YM.
Exchange execution	DXmatch evidence projected by YM	The effective working, filled, replaced or terminal state.
Financial posting	YM ledger	The resulting hold, fill and settlement posting state.

An HTTP `201` or `202` is command admission, not venue finality.

Action, outcome and price semantics

The public contract always describes the participant's selected outcome. Clients send `action`, `outcome`, and that outcome's `price_cents`:

`action`	`outcome`	Meaning
`BUY`	`YES`	Increase YES exposure at the stated YES price.
`SELL`	`YES`	Reduce YES exposure at the stated YES price.

action	outcome	Meaning
BUY	NO	Increase NO exposure at the stated NO price.
SELL	NO	Reduce NO exposure at the stated NO price.

YM derives the internal YES-book side and OPEN/CLOSE accounting effect. Do not complement NO prices: BUY NO at 35 means pay 35 cents for NO. Naked shorting is not exposed; a SELL must be covered by the named outcome position.

Prices are integer cents from 0 through 100; quantities are positive whole contracts. The public professional contract currently exposes limit orders only.

Time in force

Value	Behavior
GTC	Rest until filled, explicitly cancelled or removed by market lifecycle. Preferred for professional resting liquidity.
IOC	Attempt immediate execution and cancel any unfilled remainder. It remains a limit order with price protection.
DAY	Venue-supported session-scoped lifetime. Do not use it as a substitute for a long-lived prediction-market order.

GTD, FOK, market, stop and iceberg orders are not currently exposed.

NEW

```
text
POST /v1/orders
Authorization: Bearer <token>
Idempotency-Key: <participant-scoped-command-id>
Content-Type: application/json
```

Required body:

```
json
{
  "client_order_id": "strategy-a-000001",
  "market_id": "immutable-market-id",
  "market_generation": 7,
  "action": "BUY",
  "outcome": "YES",
  "price_cents": 45,
  "quantity": 100,
  "time_in_force": "GTC"
}
```

Do not send user_id. client_order_id is participant-defined, has a maximum length of 200 characters and must remain unique within the participant's retained order history.

Do not send the retired v1 fields side, position_side, intent, or decimal price. Unknown request fields fail closed.

Responses:

- 201: newly admitted command with a PENDING order.
- 200: exact idempotent replay of the original command.
- 4xx/5xx: inspect the canonical error and follow the retry policy.

YM transactionally records the admitted command, order, collateral reservation and durable downstream work before responding. DXmatch execution remains asynchronous.

REPLACE

```
text
POST /v1/orders/{orderID}/replace
Idempotency-Key: <new-command-id>

json
{
  "expected_order_version": 4,
  "price_cents": 46,
  "total_quantity": 120
}
```

Rules:

- Use the `order_version` from the latest canonical order snapshot.
- `price_cents` remains in the order's immutable `outcome` coordinate. A NO replacement is therefore also a NO price and must not be complemented.
- `total_quantity` is the target total for the order, including quantity already filled. It must exceed `cumulative_filled_quantity`.
- A replacement receives its own idempotency key.
- Only one mutation may be pending on an order.
- `202` means the replacement was admitted. Until `order.replaced`, the prior effective price and quantity remain authoritative.
- A rejected replacement leaves the prior effective order in place.
- Fills racing with an admitted replacement remain valid and are reflected in subsequent order versions.

DX applies an accepted REPLACE to one existing venue order as a single-order mutation. YM admission is not proof that it became effective. REPLACE does not atomically replace a bid/ask pair.

CANCEL

```
text
POST /v1/orders/{orderID}/cancel
Idempotency-Key: <new-command-id>
Content-Type: application/json

{}
```

- `202` means a venue-directed cancel was admitted and `pending_action=CANCEL`.
- `200` means cancellation is already locally effective or the command is an idempotent completed/no-op result.
- Cancellation is final only when the canonical order is terminal or an `order.cancelled` event is applied.
- Lifecycle and risk cancellations use the same canonical order state and are visible even when initiated outside the current client connection.

Idempotency

All single-order mutations require `Idempotency-Key`:

- maximum 200 characters;
- scoped to the authenticated participant;
- the `ym:` prefix is reserved;
- retry an uncertain request with the exact same key and payload;
- reusing a key with a different payload is a conflict; and
- use a distinct key for each distinct economic command.

Transport timeout is not evidence of failure. Recover by replaying the exact request or querying the resulting canonical order.

Clients must durably retain each outbound command payload under its `client_request_id`. A command result identifies admission/outcome and current order state; it is not a copy of the client's command-request journal. There is currently no general query-by-client-request-ID endpoint.

Batch commands — not enabled for the v2 pilot

```
text
POST /v1/orders/batch
POST /v1/orders/cancel-batch
```

The endpoints exist in the platform but are excluded from the approved v2 pilot until per-item idempotent-replay and error-envelope conformance passes. Do not build a pilot strategy against them yet.

The intended contract accepts at most 250 items. Each item carries its own `client_request_id` and returns its own status, acceptance and retryability.

A batch is a transport convenience, not one atomic multi-order transaction. Clients must inspect every item and retry only the exact uncertain items with the same item IDs and payloads.

Canonical reads

```
text
GET /v1/orders/{orderID}
GET /v1/orders?market_id=...&execution_status=...&cursor=...&limit=...
```

Order reads expose:

- immutable public `action` and `outcome`, with all prices and filled notional expressed in that outcome's coordinate;
- `execution_status`: `PENDING`, `WORKING`, `PARTIALLY_FILLED`, `FILLED`, `CANCELLED`, `EXPIRED` or `REJECTED`;
- `pending_action`: `REPLACE`, `CANCEL` or null;
- `state_quality`: `CONFIRMED` or `RECONCILIATION_REQUIRED`;
- original, current total, cumulative filled, remaining and working quantity;
- `order_version`; and
- admitted, venue-accepted, fill, terminal and update timestamps when known.

The single-order response supplies an `ETag` derived from `order_version` and accepts `If-None-Match`.

If `state_quality=RECONCILIATION_REQUIRED`, stop mutating that order until YM has restored confirmed venue truth.

Canonical REST command and list envelopes carry `schema_version: 2`. The single-order GET returns the bare v2 `Order` resource; its URL remains `/v1` and it does not add a second schema envelope.

Account projections

```
text
GET /v1/trades
GET /v1/positions
GET /v1/portfolio
GET /v1/accounts/{selfAccountID}/balances
```

These routes are self-scoped by the bearer token. Private fill events currently report financial `settlement_status=POSTED`; a fill is not presented as financially complete before its approved posting path succeeds.

`GET /v1/trades` is a legacy account-activity projection outside Order Contract v2: `side` and `price_cents` are in the YES-book venue coordinate, not the selected-outcome v2 order coordinate. It is useful for reconciliation but is not

the live v2 recovery authority. Use private-feed `order.fill`, whose nested order and fill price are participant- relative, for live strategy state. The legacy endpoint is deliberately omitted from the curated v2 OpenAPI.

Forward path

Persistent WebSocket order entry, post-only/maker-or-cancel, mass cancellation, cancel-on-disconnect and participant kill controls are in development. External mass quote is not part of the current contract. These additions will reuse the same command IDs, idempotency, order versions, collateral rules and private events.

---

6. Private trading feed

Status: **controlled pilot**

Current message schema: **v2**

Endpoint:

```
text
wss://ympredictions.com/v1/trading-feed
```

Required scope: `trading-feed:read`

The feed is participant-scoped and read-only. It reports canonical command, order, replacement, fill and terminal state regardless of whether the change originated from REST, lifecycle control or another authorized session for the same account.

Connection

Pass the access token in the WebSocket upgrade request:

```
text
Authorization: Bearer <access_token>
```

Participant identity is derived from the token. No `user_id`, account, MPID or venue-session selector is accepted.

WebSocket compression is disabled for the low-latency pilot profile. Clients must accept JSON text frames up to the `max_frame_bytes` announced in `stream.welcome`.

New session bootstrap

Without a cursor, the feed sends:

1. `stream.welcome`
2. `stream.snapshot_started`
3. zero or more `stream.snapshot_chunk` frames
4. `stream.snapshot_completed`
5. events committed after the snapshot high-water mark
6. `stream.ready`
7. live lifecycle events and `stream.heartbeat`

The snapshot contains all canonical nonterminal orders for the authenticated participant. Replace local nonterminal state with the completed snapshot, then apply replayed events in order. Do not begin strategy command flow before `stream.ready`.

`stream.welcome` advertises the contract name, heartbeat interval, maximum frame size and `at_least_once` delivery mode. Use those values rather than hardcoding deployment defaults.

For v2 the advertised contract is `Private Order & Execution Feed v2`, and every lifecycle and control message carries `schema_version: 2`.

Resume

Each lifecycle event includes an opaque signed `cursor`. Persist it only after the event has been validated and durably applied locally.

Reconnect with:

```
text
wss://ympredictions.com/v1/trading-feed?cursor=<URL-encoded-last-applied-cursor>
```

The server replays events after the supplied cursor, then sends `stream.ready`. A resume session does not send another account snapshot unless the client reconnects without a cursor.

The cursor is:

- opaque;
- signed by YM;
- bound to the authenticated participant and stream; and
- not a DXmatch offset or a number clients may increment.

A cursor issued by v1 is invalid for v2. Before cutover, durably consume v1 through the announced boundary and retain terminal/fill history. Then discard the v1 cursor and connect without a cursor. The v2 snapshot replaces only nonterminal order state; it does not replay v1 terminal orders or fill history.

Delivery and convergence

Delivery is at least once. A correct consumer must:

1. Require `schema_version == 2` and the expected account scope.
2. Deduplicate lifecycle facts by `event_id`.
3. Apply an order snapshot only when its `order_version` is newer than the locally stored version.
4. Persist the event and its cursor atomically in the client's own store.
5. Keep independent orders independent; do not assume one global exchange ordering across markets.

Duplicate delivery is normal recovery behavior and must not create a duplicate fill or command.

The feed is a result/state journal, not the participant's outbound request journal. Persist every submitted payload locally under `client_request_id`; pending replacements intentionally retain the prior effective order terms until DX confirms the mutation.

Lifecycle events

Event	Meaning
`order.admitted`	A canonical order was admitted by YM; it is not necessarily working at DXmatch.
`order.working`	Venue evidence confirms the order is working.
`order.replaced`	Venue evidence confirms the new single-order price/quantity is effective.
`order.fill`	A fill and complete updated order snapshot are available.
`order.cancelled`	Cancellation is effective.
`order.expired`	The venue/lifecycle made the order terminal by expiry.
`order.rejected`	The order was rejected and is terminal.
`order.command_result`	Admission/outcome for `NEW`, `REPLACE` or `CANCEL`; a rejected NEW may have no order.

Every ordinary lifecycle event carries a complete canonical order snapshot. The sole orderless case is a schema-valid `NEW` rejected before an order could exist.

Relevant event fields include:

- `event_id`, `event_type`, `cursor` and `order_version`;
- complete `order` state;
- participant-relative `action`, `outcome`, outcome price, and filled notional;
- originating command and client-request identity where applicable;
- participant-safe venue evidence: symbol, the participant's execution ID, shared trade/match ID when supplied, and venue event time;
- fill price, quantity, maker/taker role and posting state; and
- YM receive, projection and socket-send timestamps.

`order.price_cents`, `order.filled_notional_cents`, and `fill.price_cents` are all expressed in `order.outcome`. Clients never complement NO values.

Raw DXmatch replay offsets, MPID/account/session topology and the opposite participant's execution identifier are deliberately not exposed.

Control errors

The server may send `stream.error` and close the connection:

Code	Action
`INVALID_CURSOR`	Discard the cursor only after confirming the connection used the correct participant; reconnect without it for a fresh snapshot.
`CURSOR_EXPIRED`	Reconnect without the cursor and rebuild nonterminal state from a fresh snapshot.
`REPLAY_LIMIT_EXCEEDED`	Reconnect without the cursor; investigate why the client fell behind.
`SLOW_CONSUMER`	Stop command flow, improve local consumption, then resume from `last_written_cursor` or the client's earlier safely applied cursor.
`STREAM_ERROR`	Preserve the last safely applied cursor and reconnect with bounded backoff. Escalate persistent failures.

`last_written_cursor` means the last event written to the socket, not necessarily the last event your application applied. Prefer your own durable last-applied cursor.

Heartbeats and token rotation

Treat absence of both data and heartbeats beyond the announced interval plus a small network allowance as a disconnected session. Refresh the short-lived OAuth token out of band, open a replacement connection and resume from the last applied cursor.

Client-to-server application messages are not part of Private Feed v2. Standard WebSocket ping/pong may be used by the transport, but it does not acknowledge business events.

Forward path

The pilot feed already provides participant isolation, snapshot, replay and resume. Before contractual general availability, YM intends to complete multi-instance/high-availability evidence, retention and disaster-recovery proof, sustained slow-consumer testing and a published availability target. Future command WebSockets will share this canonical event stream rather than replace it.

---

7. Errors and retry policy

Status: trading-core canonical error envelope **available**; gateway-edge envelope and public versioned error registry **in development**

Canonical order errors use this shape:

```
json
{
  "code": "stale_market_generation",
  "message": "market generation is stale",
  "retryable": false,
  "command_id": null,
  "client_request_id": "strategy-a-new-42",
  "details": {}
}
```

The deprecated `error` field may duplicate `message` during transition. New clients should use `code`, `message` and `retryable`. Authentication, authorization, gateway rate-limit, and upstream-availability failures may still arrive as the transport-edge shape `{"error":"..."}`. Treat those as HTTP transport failures; do not infer a command outcome from that body.

HTTP policy

Status	General meaning	Client action
200	Read success, exact idempotent replay, or locally completed/no-op mutation.	Inspect the canonical command and order.
201	A new order command was durably admitted.	Wait for exchange-effective state.
202	A replace or venue-directed cancel was durably admitted.	Wait for the corresponding effective event.
304	The order version matches `If-None-Match`.	Keep the local snapshot.
400	Invalid request or unsupported contract value.	Correct the request; do not blind-retry.
401	Missing, invalid or expired token.	Obtain a valid token, then retry with the same command ID and payload.
403	Scope, account or policy denial.	Do not retry until authorization/policy changes.
404	Resource not found in the participant scope.	Reconcile identity; do not create a replacement identity blindly.
409	Current state conflicts with the command.	Follow the specific code below.
429	Rate limit exceeded.	Honor `Retry-After`; preserve command identity.
503	Required admission, reconciliation or rate-limit dependency unavailable.	Follow `retryable`; use the same ID for an uncertain command.

Important validation codes

These require a corrected request rather than a retry:

- `missing_idempotency_key`
- `invalid_idempotency_key`
- `reserved_idempotency_key`
- `invalid_json`
- `invalid_canonical_order`
- `invalid_order`
- `invalid_order_id`
- `invalid_replace`

Identity and idempotency conflicts

Code	Meaning	Action
`idempotency_payload_conflict`	The NEW key already identifies a different payload.	Generate a new key only for a genuinely new command.
`replace_command_conflict` / `cancel_command_conflict`	The mutation key conflicts with an existing command payload.	Reconcile the original command; do not overwrite its identity.
`client_order_id_conflict`	The participant has retained that client order ID already.	Reuse it only to recover the original order; otherwise choose a new order ID.
`command_already_pending`	Another mutation is awaiting effective venue state.	Wait for the private-feed outcome, refresh the order, then decide again.

Market and quote conflicts

Code	Meaning	Action
`stale_market_generation`	The command used an old admission generation.	Refresh the canonical market and re-evaluate the command.
`market_not_open`	Lifecycle no longer accepts this command.	Stop sending until a later canonical state explicitly permits it.
`market_temporarily_unavailable`	Admission/readiness is currently blocked.	Refresh state; retry only if the same command remains valid.
`market_quote_unavailable`	The protected IOC quote could not be verified and the command was terminally rejected.	Refresh the book and make a new economic decision under a new ID; replaying the old ID only recovers that rejection.
`market_quote_no_liquidity`	No executable protected liquidity exists.	Wait for book change or submit a different limit command with a new ID.
`market_quote_moved`	The protected quote changed before admission.	Refresh and consciously reprice using a new command identity.
`market_quote_slippage_exceeded`	Available execution exceeded the supplied protection.	Re-evaluate price protection; never auto-widen without strategy authority.

Replace conflicts

Code	Meaning	Action
`order_not_replaceable`	The order is no longer in a replaceable working state.	Refresh the order.
`order_version_conflict`	`expected_order_version` is stale.	Refresh and decide against the latest version.
`replace_quantity_filled`	Target total does not exceed cumulative fills.	Refresh fill state and choose a valid target.
`replace_noop`	Target price and remaining quantity are already effective.	Treat as no change; do not loop.
`insufficient_replace_reserve`	Additional collateral/exposure is unavailable.	Reduce/cancel or fund the account; do not blind-retry.
`short_quantity_increase_not_supported`	This BUY NO order's total quantity cannot currently be increased safely.	Cancel it and submit a separately collateralized NEW if still intended.

Coverage and participant policy

- `insufficient_coverage` means available cash or position coverage is insufficient at admission.
- `money_gate_closed`, `user_not_eligible` and `responsible_gambling_*` are policy decisions, not transient venue failures.
- `order_admission_busy` may be retried with bounded jitter and the same exact ID/payload.

Uncertain outcomes

`command_commit_unknown` is the critical recovery case: YM could not prove to the caller whether the command transaction committed. Retry the exact command with the same `Idempotency-Key`. Do not generate a new key.

`reconciliation_required` means YM will not return or mutate uncertain canonical state. Stop mutation, preserve the last confirmed state and retry the read with bounded backoff. Escalate if it persists.

For connection loss or any HTTP timeout after request transmission:

1. Do not infer failure.
2. Retry the exact payload with the same key.
3. Query orders and consume the private feed.
4. Create a new command ID only after the original outcome is known and a new economic decision is made.

If a response or feed event contains a canonical command with `admission_status=REJECTED`, that identity is terminal even when its transport status was `503`. Replaying it recovers the same rejection; it does not rerun admission.

Backoff

Use bounded exponential backoff with jitter for retryable transport and `5xx` failures. Honor `Retry-After` on `429`. Cancellation and risk-reducing traffic should use the separately provisioned cancellation capacity but must not spin without backpressure.

Forward path

YM intends to publish a generated, versioned registry containing every external error code, retry class and remediation, then enforce it against handler tests. Until that is complete, the curated OpenAPI plus this policy define the pilot contract; unknown codes must fail safe and be reported with correlation data.

---

8. Market data

Status: REST snapshots **available**; consumer WebSocket **preview**; professional recoverable stream **in development**

YM currently supports public REST snapshots and recent-trade reads suitable for discovery, command preflight and recovery checks. The existing consumer WebSocket is not yet the professional market-data contract.

L2 book snapshot

```text
GET /v1/book?market_id={marketID}&depth={1..200}
GET /v1/markets/{marketID}/book?depth={1..200}
```

Depth defaults to 20 and is capped at 200 levels per side.

Example:

```json
{
  "market_id": "immutable-market-id",
  "depth": 20,
  "bids": [
    {"price": 0.44, "price_cents": 44, "qty": 120}
  ],
  "asks": [
    {"price": 0.46, "price_cents": 46, "qty": 80}
  ],
  "seq": 184203,
  "updated_at": "2026-09-01T12:00:00Z"
}
```

Use `price_cents` when present. Decimal `price` exists for consumer compatibility.

`seq` is a monotonic YM market-data projection sequence for the running projection, not a durable professional replay cursor and not a DXmatch source sequence. `updated_at` is YM's projection timestamp; it must not be interpreted as an exchange-origin timestamp.

A closed or resolved market returns an empty book. An unknown/uninitialized market may also return an empty snapshot; always combine the book with canonical market lifecycle state.

Recent public trades

```
text
GET /v1/markets/{marketID}/trades?limit=200
```

This returns recent taker-side public trade activity with pseudonymous participants, side, liquidity role, price, quantity and occurrence time. It is not a participant's private drop copy; use Private Feed v2 for canonical own- order and fill recovery.

The L2 book and public trade tape use the common YES-book price coordinate. They do not switch coordinates for a participant's selected outcome.

The exact bridge is:

- YES ask at `p` corresponds to `BUY YES @ p` or `SELL NO @ (100-p)`.
- YES bid at `p` corresponds to `SELL YES @ p` or `BUY NO @ (100-p)`.

This complement is a market-data interpretation only. Public order commands always send their selected outcome price directly; clients must not complement NO order prices before submission.

Statistics

```
text
GET /v1/markets/{marketID}/stats
```

Statistics include 24-hour and total volume, trade/trader counts, average position and open interest. These are derived analytics and are not part of order admission or exchange-effective state.

Reference anchors

Provider anchors and price history are product/reference signals, not the YM order book and not executable prices. A strategy must not treat an anchor as a firm quote.

Consumer WebSocket preview

The existing endpoint is:

```
text
/v1/ws?markets=<csv>&streams=book_deltas,trades,ticker
```

It is intentionally not included in the market-maker OpenAPI. Its current sequence is local to a market-data service process, it does not provide the professional replay contract, and slow-consumer behavior is not suitable for market-making recovery. Its browser-oriented connection policy also differs from the headless private trading feed.

Do not build production market-making state solely from this preview.

Required client behavior today

- Discover immutable market IDs and generations through canonical catalog REST.
- Obtain an L2 snapshot before quoting.

- Re-fetch a snapshot after any suspected gap, reconnect or stale local state.
- Stop quoting when canonical lifecycle/admission no longer permits trading.
- Use private execution events, not public trade inference, for own-order truth.
- Bound snapshot polling; the pilot REST surface is not a substitute for the professional streaming transport under development.

Forward path

The professional market-data transport will be based on DXmatch-derived full book authority with bounded subscriptions, explicit fresh/stale state, coherent snapshot recovery, generation/lifecycle fencing and measured capacity. YM is evaluating the appropriate DX-native transport and will not manufacture replay guarantees the venue feed cannot support. The consumer WebSocket will either be upgraded behind an explicit professional contract or remain a separate retail presentation feed.

9. Limits, availability and roadmap

Status: **controlled pilot policy; not a public SLA**

Limits protect participant fairness, financial correctness and downstream venue capacity. They are keyed to authenticated identity where possible; a source-IP limit remains only a coarse pre-authentication denial-of-service boundary.

Current protocol limits

Resource	Current limit
Idempotency key	200 characters; `ym:` prefix reserved
Client order ID	200 characters
Batch NEW	Intended maximum 250 items; not enabled for v2 pilot
Batch CANCEL	Intended maximum 250 items; not enabled for v2 pilot
Canonical order page	500 items maximum
Canonical market page	500 items maximum
Recent public market trades	200 items maximum
REST L2 depth	200 levels per side; default 20
Private-feed frame	Server advertises `max_frame_bytes`; current profile caps at 256 KiB
Snapshot chunk	At most 100 orders, and may be smaller to respect frame size

The server response is authoritative if a deployment advertises a narrower limit.

Current professional request profile

The controlled benchmark/onboarding profile is presently configured at 3,300 authenticated requests per minute. Professional cancel traffic uses an independent bucket from other professional requests, with the same current configured ceiling.

This value was selected to test a 50-command/second workload with control headroom. It is not an unlimited tier, a guaranteed sustained rate or a contractual SLA. A participant must honor `429` and `Retry-After`.

The pre-authentication IP limit is not participant identity. Approved clients must not assume that distributing requests across addresses increases their entitlement.

Availability matrix

Capability	State	Qualification
Canonical catalog REST	Available	Public discovery; canonical view required for trading.
L2 snapshot and public trade REST	Available	Snapshot/recovery reads, not a streaming SLA.
Canonical REST commands	Pilot	Approved accounts only.
Canonical account reads	Pilot	Self-scoped by bearer identity.
Private execution feed	Pilot	Resumable at-least-once delivery; contractual HA/SLO not yet published.
Batch commands	In development	Excluded until per-item replay and error-envelope conformance passes.
Consumer market-data WebSocket	Preview	Not approved as a professional recovery source.
Persistent command WebSocket	In development	Will reuse the canonical command path.
Recoverable professional market data	In development	DXmatch-derived snapshot/recovery design.
Mass cancel/cancel-on-disconnect/kill controls	In development	Required before broad external onboarding.
FIX	In development	Conditional on vendor access, conformance and measurable benefit.
External quote-set/mass-quote	Not offered	Internal Paper-LP routes are not an external contract.

Latency and capacity

YM measures end-to-end command latency from client request through canonical exchange-effective private-feed delivery, including p50, p95, p99, p99.9 and maximum observations. Internal benchmark results are evidence used to size the pilot; they are not an external guarantee until a workload, percentile, measurement boundary, exclusion policy and remedy are contractually published.

Capacity qualification must cover:

- NEW, REPLACE, CANCEL, full fills and partial fills;
- same-order, same-account, same-market and distributed-account contention;
- reconnect/replay, slow consumers and token rotation;
- cancel storms and lifecycle drains;
- one participant at limit without harming another; and
- the actual Azure-to-DXmatch network floor.

Current onboarding constraints

- Credentials are provisioned manually.
- One pilot confidential client maps to one dedicated account.
- Scopes and professional rate entitlement are explicitly allowlisted.

- Private-feed access is explicitly enabled.
- Environment, economic terms, allowed markets and support procedures are supplied during onboarding.
- There is no self-service account/strategy entitlement or published general- availability support policy yet.

Roadmap to professional availability

Before expanding beyond controlled participants, YM intends to complete:

1. Sustained release-grade command/fill benchmarks on exact deployed images.
2. Persistent WebSocket order entry through the canonical command path.
3. Recoverable professional market data.
4. Mass cancel, cancel-on-disconnect and participant kill controls.
5. Firm/application/strategy/account entitlement and credential lifecycle.
6. Connection, subscription and outstanding-order quotas.
7. Multi-instance private-feed and disaster-recovery evidence.
8. A public status surface, changelog, conformance suite and support process.
9. Published SLO and capacity terms based on measured workload and DXmatch tenant capacity.

These are forward-looking integration notes, not current commitments or dates. Capabilities remain unavailable until their status is changed in a versioned participant contract.