



TECHNICAL HANDBOOK

DXtrade API

A practical guide to REST, Push, troubleshooting, and safe integration

For Velotrade challenge-account traders and their development assistants

VERSION 2.11

Core flows verified 28 July; release updated 4 August 2026

Contents

1. Preface: purpose and responsibility
2. Canonical connection, authentication, secrets, and dependencies
3. Getting started and safety invariants
4. Mutation gate, REST, and reconciliation
5. Push API guide
6. Asset mapping and response normalisation
7. Troubleshooting and canonical errors
8. Verification evidence and live policy check
9. Source references
10. Important disclaimer

Security note: never place credentials or session tokens in this document, source control, screenshots, logs, or AI prompts.

Preface: purpose and responsibility

This handbook is provided by Velotrade solely to facilitate the technical implementation of DXtrade REST and Push connectivity. It is practical, informational guidance—not a prescribed implementation, trading strategy, compliance determination, or representation that any particular design, control, test, or workflow is required, complete, suitable, or error-free.

Velotrade does not currently support or provision FIX API connectivity. Published DXtrade FIX specifications are reference material only and do not establish availability or entitlement on Velotrade accounts.

The trader and each person or organisation implementing an integration remain responsible for deciding how the integration should operate, testing it for their intended use, supervising it, securing credentials, controlling risk, and ensuring that its use complies with the current Velotrade website terms, rules, disclosures, product information, account conditions, and applicable law.

Velotrade's official website is the live source of truth for customer-facing policies and product conditions. Those pages may change after this handbook is published. Before enabling any action that can create, modify, cancel, or close an order or position, retrieve and review the current official pages listed in API safety and live policy check (API_SAFETY_AND_LIVE_POLICY.md). If the live pages cannot be accessed or their application is uncertain, keep the integration read-only and seek clarification.

The DXtrade platform and its interfaces may also change. Runtime discovery, defensive reconciliation, account-specific validation, and independent professional advice where appropriate remain the implementer's responsibility.

Canonical connection values

This file is the canonical location for Velotrade DXtrade connection values. Other guides should link here rather than maintaining another independent copy.

Setting	Value	Status
REST base	https://dx.velotrade.com/dxsca-web	Live-verified
REST login path	/login	Live-verified
REST login domain	default	Live-verified
REST authorisation	Authorization: DXAPI <sessionToken>	Live-verified
Business Push	wss://dx.velotrade.com/dxsca-web/?format=JSON	Live-verified; trailing slash required
Market-data Push	wss://dx.velotrade.com/dxsca-web/md?format=JSON	Live-verified

The website login's `vendor=velotrade` parameter is not the REST login domain. Do not substitute it for `default`.

Canonical authentication guidance

Use POST /login at the REST base in CONNECTION_VALUES.md with only:

```
{
  "username": "<USERNAME>",
  "domain": "default",
  "password": "<PASSWORD>"
}
```

A successful response contains `sessionToken` and `timeout`. Send the token as `Authorization: DXAPI <sessionToken>`. Do not log it. Use POST /ping only at a conservative interval and call POST /logout during shutdown. A successful logout may be HTTP 200 with an empty body.

The documented login request has no TOTP, backup-code, security-code, or MFA continuation field. If login returns HTTP 500/error 110, stop retries and complete any required password/MFA flow through the web interface. Do not claim that code 110 identifies the exact interactive step.

Platform-level HMAC availability does not prove that a customer principal is provisioned and is not an MFA bypass.

Secret handling and redacted output

Approved environment variables are documented in the example `.env.example` files. Never commit `.env`; the package-root `.gitignore` excludes it.

Prefer a secret manager or silent terminal entry. Example for a temporary shell session:

```
IFS= read -rs VELOTRADE_USERNAME  
IFS= read -rs VELOTRADE_PASSWORD  
export VELOTRADE_USERNAME VELOTRADE_PASSWORD
```

Do not place credentials directly in shell history, source files, URLs, AI prompts, screenshots, or reports. Do not print session tokens or `Authorization` headers.

Examples redact account codes, owners, internal identifiers, and token-like fields by default. `--unsafe-full-output` is an explicit local-only diagnostic override. Output produced with that switch must not be pasted into chat, attached to support, or committed without a separate redaction pass.

Log out REST tokens in `finally`, explicitly close Push subscriptions and sockets, and rotate credentials after accidental disclosure. For an incident:

1. stop the affected process;
2. log out or revoke the active token where possible;
3. rotate exposed credentials through the authorised account workflow;
4. remove the material from logs, history, tickets, and shared storage;
5. record what was exposed and notify the authorised operator or Velotrade.

Dependencies and architecture diagnostics

The documentation and source are architecture-neutral. Do not maintain separate Intel and Apple Silicon KB editions or copy virtual environments between machines.

Run the doctor first

Node.js:

```
cd examples/node
npm install
npm run doctor
npm run smoke
```

Python:

```
cd examples/python
python3 -m venv .venv
source .venv/bin/activate
python -m pip install -r requirements.txt
python doctor.py
python smoke.py
```

On Windows, activate with `.venv\Scripts\Activate.ps1`.

The doctor reports operating system, CPU architecture, interpreter bitness, runtime versions, missing modules, and detectable mixed-architecture binary packages. Create the virtual environment with the same interpreter that will run the examples.

`websocket-client` is imported in Python as `websocket`; it is not the different `websockets` package. Node 20 uses the pinned `ws` dependency. A modern runtime may provide native `fetch`, but the examples keep declared dependencies explicit and reproducible.

`npm run check` proves syntax plus offline normaliser fixtures. `npm run smoke` proves runtime imports/helpers are available without contacting Velotrade. Node 22 may use native platform capabilities, while Node 20 uses the pinned `ws` package for WebSockets. Outbound network access may require separate authorisation in managed execution environments.

Direct Python dependencies are exactly pinned in `requirements.txt`. Wheel hashes vary by operating system and architecture; do not publish one platform's hash as universal. If offline installation is required, provide separately versioned and signed wheelhouses for each supported platform/architecture and verify them before use.

Dependency inventory is in `integrity/SBOM.spdx.json`; package hashes are in `integrity/CHECKSUMS_SHA256.txt`.

Getting started

This is the shortest safe route from no integration to a working, read-only Velotrade DXtrade connection.

What you need

- A Velotrade challenge-account username and password.
- Node.js 20 or later, or Python 3.10 or later.
- The example files from this documentation package.
- Access to Velotrade's current official website pages for the mandatory pre-mutation check in API safety and live policy check (API_SAFETY_AND_LIVE_POLICY.md).

Your trading-account username and password are also used for DXtrade API token authentication. Do not paste them into source code or an AI conversation.

Use REST and Push for Velotrade client integrations. FIX Trading and FIX Market Data are not currently supported or provisioned by Velotrade. The existence of generic DXtrade FIX specifications does not provide a usable Velotrade FIX gateway or account entitlement.

Password-based REST login works only when the underlying account can complete authentication without additional interactive steps. If the password has expired or MFA enrolment/challenge is pending, complete the password change and MFA flow through the Velotrade web interface before retrying REST login. The REST `/login` contract has no TOTP, backup-code, or MFA-continuation field.

Velotrade connection values

Setting	Value	Status
REST base URL	<code>https://dx.velotrade.com/dxsca-web</code>	Verified
REST login domain	<code>default</code>	Verified
Business Push URI	<code>wss://dx.velotrade.com/dxsca-web/?format=JSON</code>	Verified and intended to remain stable
Market-data Push URI	<code>wss://dx.velotrade.com/dxsca-web/md?format=JSON</code>	Verified and intended to remain stable
REST token header	<code>Authorization: DXAPI <sessionToken></code>	Verified

The website login's `vendor=velotrade` setting is unrelated to the REST login's `domain`. Using `velotrade` as the REST domain failed authentication; use `default`.

Ask for account context first

Before requesting credentials or beginning discovery, ask the authorised operator for the account type or challenge plan, current phase, applicable phase-specific restrictions/objectives, and whether the account is newly issued or already contains activity. Record these as operator-supplied facts; do not infer them from balance, account name, account number, or API behaviour. If later API discovery or current official Velotrade material conflicts with the statement, stop mutations and request clarification.

Five-step connection check

1. Store credentials locally

Copy the relevant `.env.example` to `.env` and fill in the username and password. The supplied `.gitignore` excludes `.env`.

Never commit the `.env` file.

2. Log in

Send:

```
POST https://dx.velotrade.com/dxsca-web/login
Content-Type: application/json

{
  "username": "<USERNAME>",
  "domain": "default",
  "password": "<PASSWORD>"
}
```

A successful response contains `sessionToken` and `timeout`. The tested deployment returned an inactivity timeout of `00:30:00`.

If this documented request returns HTTP 500 with error code 110, stop retrying and check the account's web-login state. DXtrade confirmed on 29 July 2026 that an expired password or pending MFA step can produce a non-SESSION/non-REJECT authentication result which `/login` maps to this generic server error. Remediate the password/MFA state through the web UI, then retry once. Do not add guessed `totp`, `securityCode`, or `backup-code` fields to the REST body.

3. Discover the API account code

Request the accessible user list with the token:

```
GET https://dx.velotrade.com/dxsca-web/users
Authorization: DXAPI <SESSION_TOKEN>
```

Inspect each returned account:

- choose the intended account;

- confirm `accountStatus` permits the required action;
- read the full `account` field.

Do not assume a single fixed response wrapper. The tested deployment returned `account` records beneath a `userDetails` structure, while generic DXtrade examples may show `users/accounts` arrays directly. Traverse recognised containers or recursively collect objects that contain both `account` and `accountStatus`; then deduplicate by the full account code.

The full code has two parts, for example:

```
default:<ACCOUNT_NUMBER>
```

Do not build this value by assumption. Discover it on every new environment. An older or disabled account may also appear.

When the code is used in a URL path, encode the colon:

```
default%3A<ACCOUNT_NUMBER>
```

4. Run read-only checks

Before considering any trading action, verify:

```
GET /accounts/{encodedAccountCode}/metrics
GET /accounts/{encodedAccountCode}/portfolio
GET /accounts/{encodedAccountCode}/positions
GET /accounts/{encodedAccountCode}/orders
GET /accounts/{encodedAccountCode}/instruments/query
```

Confirm:

- the account is the one you intended;
- currency and equity are plausible;
- existing positions and orders are understood;
- the instrument symbol, trading status, minimum size, increment, and maximum size were discovered from the account-specific endpoint.

5. Log out

When the program is finished:

```
POST https://dx.velotrade.com/dxsca-web/logout
Authorization: DXAPI <SESSION_TOKEN>
```

Always attempt logout in a `finally` block so errors do not leave tokens active. Treat an empty HTTP 200 logout body as success; do not parse an empty body as JSON. A post-logout ping should fail with HTTP 401.

Run the examples

Node.js:

```
cd examples/node
cp .env.example .env
npm install
npm run doctor
npm run smoke
npm run rest
npm run multi-asset
```

Python on macOS or Linux:

```
cd examples/python
python3 -m venv .venv
source .venv/bin/activate
python -m pip install -r requirements.txt
cp .env.example .env
python doctor.py
python smoke.py
python rest_readonly.py
```

Python on Windows PowerShell:

```
cd examples\python
py -m venv .venv
.\venv\Scripts\Activate.ps1
python -m pip install -r requirements.txt
Copy-Item .env.example .env
python doctor.py
python smoke.py
python rest_readonly.py
```

Expected outcome:

- login accepted;
- one or more accounts listed;
- the selected full account code printed;
- metrics, positions, orders, and matching instruments summarised;
- logout accepted.

`npm run multi-asset` remains read-only. It enumerates the complete account-specific catalogue and reports eligible `FULL` candidates across every discovered asset class, including Crypto where available. Its output is discovery evidence, not a recommendation or automatic mutation selection.

To diagnose a submitted order after retaining its returned internal ID:

```
export VELOTRADE_ORDER_ID=<RETURNED_ORDER_ID>
npm run history
```

This read-only lookup queries `/orders/history` and prints a redacted summary of final status, executions, filled quantity, and any `rejectReason`.

All examples use redacted structured output by default and never print the password or session token. Full raw output requires the explicit `VELOTRADE_UNSAFE_FULL_OUTPUT=true` debug switch and must be reviewed before distribution.

The Node controlled lifecycle example is disabled by default. Its normal run is a dry-run plan; enabling any mutation requires `--execute`, every explicit acknowledgement/input flag documented in `examples/README.md`, and a completed live mutation gate.

Fresh-account acceptance check

Use this sequence before enabling any trading logic on a newly issued account:

1. Record the operator-supplied account type or challenge plan, current phase, and any phase-specific restrictions/objectives before requesting credentials.
2. Start with `VELOTRADE_ACCOUNT_CODE` unset.
3. Run the read-only REST example and record every discovered account's full code, status, currency, and position mode.
4. Set `VELOTRADE_ACCOUNT_CODE` to the intended full `clearing:account` value.
5. Re-run REST discovery and confirm zero or fully understood starting positions/orders.
6. Set `VELOTRADE_SYMBOL_SEARCH` to a symbol family you expect, but accept only account-specific results.
7. Run the read-only Push example. Confirm one `AccountPortfolios` response, one `MarketData` response, correct `inReplyTo`, and Push ping handling.
8. Stop the examples and verify REST logout.
9. Before a mutation test, reconfirm the account type/phase and state the account, symbol, side, type, minimum quantity, protection, peak-margin ceiling, realised-loss ceiling, and flat-shutdown rule.
10. Use one minimum-size pending order first: create it away from market, reconcile it, modify it without `type`, cancel it, and verify it is gone.
11. For a position test, open once, retrieve the `positionCode`, attach and confirm a full closing STOP without `quantity`, then close the exact position and verify zero positions/orders.

Stop immediately if discovery is ambiguous, the account is not `FULL_TRADING`, an instrument is not `FULL`, a required size/margin fact is missing, PUSH becomes stale, or protection cannot be confirmed.

Before adding trading logic

Read the complete REST guide and troubleshooting guide. In particular:

- generate a unique `orderCode` for every new order;
- never retry a trading request merely because the client timed out;
- discover symbols and size rules rather than hard-coding them;
- retrieve the actual position before closing it;
- verify that no unintended working orders or positions remain;
- complete the live website gate in API safety and live policy check (API_SAFETY_AND_LIVE_POLICY.md).

Canonical safety invariants

These are the non-negotiable technical invariants for examples and controlled tests.

1. Default to read-only and redact output by default.
2. Keep credentials and session tokens out of source, logs, URLs, screenshots, reports, and AI prompts.
3. Discover the exact account and account-specific instrument facts at runtime.
4. Require an explicit bounded mandate before mutations.
5. Complete the fresh mutation policy gate immediately before mutations.
6. Generate unique identifiers and never blindly retry a trading POST.
7. Include an explicit compatible tif on every new order. The controlled harness uses the live-verified GTC value for MARKET opens/closes and STOPS.
8. Reconcile acknowledgements through positions, orders, history, portfolio, and Push where available.
9. Confirm a full position-specific closing STOP before treating a position as protected; omit quantity for that full closing STOP/LIMIT on the tested deployment.
10. Apply margin and loss ceilings to aggregate exposure.
11. Stop on stale, conflicting, unavailable, or ambiguous state.
12. Planned shutdown must prove the authorised final position/order state, explicitly close Push subscriptions, close sockets, and log out REST.
13. Technical API success does not establish policy compliance.

Mutation gate

This gate applies immediately before any request that can create, modify, cancel, or close an order or position. Strictly read-only connectivity does not require mutation permission, but still requires secure credentials, discovery, redacted output, and understood starting state.

1. Record the mandate

Reconfirm the account type or challenge plan and current phase that the operator supplied at the beginning of the session, together with any phase-specific restrictions or objectives and whether pre-existing activity was expected. If the API or a current official page exposes a conflicting fact, stop mutations and request clarification. Do not infer or silently replace the operator-supplied context.

Record the authorised account, duration, permitted actions and symbols or asset classes, minimum-size rule, aggregate margin ceiling, realised-loss ceiling, protection requirement, and shutdown condition. If the API does not clearly identify the challenge plan or phase, retain the operator's statement as the session context. Do not infer a plan from account size or name.

2. Review current official Velotrade pages

Open the named pages directly or navigate to them with an ordinary interactive browser. Do not crawl, scrape, harvest, or recursively enumerate the site. Record only URL, UTC retrieval time, applicability, and the decision; do not copy policy text into this package.

If dynamically rendered content is incomplete:

1. open it in an interactive browser;
2. use an official downloadable addendum if the page provides one;
3. ask the authorised operator;
4. contact Velotrade support.

Remain read-only while required content is unavailable or ambiguous.

3. Check instrument-specific event facts

Velotrade's current pages determine which events and restrictions matter. If an applicable current policy requires an issuer event date that Velotrade does not publish, an issuer's official investor-relations or regulatory filing page is an acceptable primary source for the factual date only. Record the issuer URL, event type, published time zone, UTC retrieval time, and decision.

An issuer source cannot override or interpret Velotrade policy. If earnings, dividend, ex-date, corporate-action, or time-zone evidence remains unclear, do not mutate that equity and ask the operator or Velotrade.

4. Reconcile website and API catalogues

Use the mapping and mismatch rules in `ASSET_CLASS_AND_CATALOGUE.md`. A symbol selected for mutation must be clearly supported by account-specific API discovery and the current customer-facing product information, or the discrepancy must be clarified first.

5. Reconfirm technical state

Immediately before mutation, refresh account status, positions, orders, instrument status, quantity facts, quote, margin information, and protection plan. Then apply `SAFETY_INVARIANTS.md` and `RECONCILIATION.md`.

Velotrade DXtrade REST API guide

Status: authentication, discovery, read-only calls, pending-order lifecycle, protected position lifecycles, concurrent positions, multi-asset behaviour, and shutdown were verified through 29 July 2026.

REST is the action and snapshot API

Use REST to:

- authenticate and log out;
- discover users, accounts, and instruments;
- request current account and portfolio state;
- request quotes or candles;
- place, modify, and cancel orders.

REST works without a WebSocket connection. Push is recommended for a long-running algorithm because it delivers live updates, but it is not a prerequisite for a REST request.

Authentication

Base URL:

```
https://dx.velotrade.com/dxsca-web
```

Login:

```
{
  "username": "<USERNAME>",
  "domain": "default",
  "password": "<PASSWORD>"
}
```

Send the returned token on subsequent requests:

```
Authorization: DXAPI <SESSION_TOKEN>
```

POST /ping renews the token's inactivity timeout. POST /logout invalidates it. Do not log tokens.

Password expiry and MFA

POST /login supports only username, domain, and password. It does not support a documented TOTP, backup-code, MFA-continuation, or other multi-step authentication field.

DXtrade confirmed on 29 July 2026 that the underlying authentication service may return a state other than a clean SESSION or REJECT when a password has expired or MFA enrolment/challenge

is pending. The REST endpoint currently maps such states to HTTP 500, error code 110, instead of a distinct actionable authentication response.

When this occurs:

1. stop automated login retries;
2. do not guess additional JSON fields;
3. complete the password change and MFA enrolment through the web interface;
4. retry the documented REST login once the account is no longer in the expired/MFA-pending state.

DXtrade is still reviewing the definitive error mapping for expired password, required MFA enrolment, required MFA challenge, and incorrect credentials. Until that reference is supplied, document HTTP 500/110 as known authentication-state mapping behaviour, not proof of a server outage and not a precise diagnosis of which interactive step is pending.

HMAC status

HMAC request signing is enabled at the Velotrade platform level, but this does not establish that a customer-specific HMAC principal can be issued for a demo or challenge account. DXtrade explained that an HMAC principal authenticates against a configured service username/password; it is therefore not inherently independent of password expiry or pending MFA and must not be described as an MFA bypass.

Customer-specific provisioning, permissions, Push compatibility, lifecycle, and independence from web-login state remain pending confirmation. Do not present HMAC as part of the beginner/customer workflow unless Velotrade and DXtrade explicitly provision and document it for that account class.

Do not assume every successful response contains JSON. The deployed logout endpoint can return HTTP 200 with an empty body. Check status, content type, and whether the body is empty before parsing. Live testing confirmed that a subsequent POST /ping with the logged-out token returns HTTP 401, error 1, Authorization required.

Account discovery is mandatory

Call GET /users, then choose the intended account from the returned accounts collections.

Check:

- account: complete clearing-and-account code;
- accountStatus: FULL_TRADING, CLOSE_ONLY, NO_TRADING, TERMINATED, or EXPIRED;
- baseCurrency;
- isPositionBased.

The tested user had both an older `NO_TRADING` account and a current `FULL_TRADING` account. Selecting the first account blindly would have been wrong.

Use the complete code in API messages. URL-encode it when placed in a path. Supplying only the visible account number produced HTML 404 responses.

Instrument discovery is mandatory

General instrument search:

```
GET /instruments/query?symbols=<SEARCH>
```

Account-specific details:

```
GET /accounts/{encodedAccountCode}/instruments/query?symbols=<SEARCH>
```

Use the account-specific result before building an order. Check:

- `symbol`;
- `tradingStatus`;
- `minOrderSize`;
- `maxOrderSize`;
- `minOrderSizeIncrement`;
- `marginRate`;
- `assetClass`.

Prefer an explicit price increment or tick from current instrument metadata. Fresh-account discovery supplied size and margin data but no explicit tick. Parsed JSON numeric values are not authoritative for displayed precision: 1.2300 may become 1.23. If no tick exists, retain a text-preserving raw account-authorized quote representation and use decimal-safe helpers. A safely distant pending LIMIT validation is itself a mutation; complete the mutation gate and obtain bounded authorisation before submitting it. Quote formatting remains observed evidence, not a permanent tick-size contract.

The tested Bitcoin symbol was `BTCUSD`; the public starter page's `BTC/USD` symbol did not match this account. A later account-wide discovery exposed `FULL` equity, Forex, commodity, index/ETF, and crypto instruments. Apply the website/API mapping and mismatch procedure in Asset class and catalogue (`ASSET_CLASS_AND_CATALOGUE.md`) before selecting a mutation symbol.

Account-specific examples included `AAOI 0.01`, `EURUSD 0.01`, `XAU 0.01`, `SPY 0.01`, and `BTCUSD 0.001`. These are examples, not universal constants.

Quantity semantics are asset-specific. Forex discovery expressed minimum, increment, and maximum in lots, while the REST order request required base-currency units. On two tested EURUSD accounts, the working mapping was:

```
REST Forex quantity = discovered lots × 100,000
0.01 lot = 1,000 base units
```

The tested EURUSD payload therefore required `quantity: "1000"`, not `"0.01"`. This is deployment evidence, not a universal contract specification. Confirm current official instrument information and account responses before applying the mapping. Do not apply it to other asset classes. Do not estimate every asset with one formula; prefer complete contract specifications or a broker margin preview and verify live account metrics.

Valid size rule:

```
quantity >= minOrderSize
and
(quantity - minOrderSize) is an exact multiple of minOrderSizeIncrement
and
quantity <= maxOrderSize when a finite maximum applies
```

Use decimal arithmetic rather than binary floating-point equality for this calculation.

Read-only account endpoints

Purpose	Endpoint
Metrics	GET /accounts/{account}/metrics
Metrics including positions	GET /accounts/{account}/metrics?include-positions=true
Portfolio snapshot	GET /accounts/{account}/portfolio
Open positions	GET /accounts/{account}/positions
Working orders	GET /accounts/{account}/orders
Account instruments	GET /accounts/{account}/instruments/query

The portfolio contains current positions and working orders. It is not a complete historical record.

One-off market data

```
POST /marketdata
```

```
{
  "account": "default:<ACCOUNT_NUMBER>",
  "symbols": ["<DISCOVERED_SYMBOL>"],
  "eventTypes": [
    {
      "type": "Quote",
      "format": "COMPACT"
    }
  ]
}
```

Use Push for continuous quotes rather than polling this endpoint rapidly.

Order request fields

Important single-order fields:

Field	Meaning
orderCode	Client-generated identifier unique on the account
type	MARKET, LIMIT, or STOP
instrument	Discovered DXtrade symbol
quantity	Discovered instrument quantity scale; do not assume universal units or lots
positionEffect	OPEN or CLOSE on position-based accounts
positionCode	Required when closing a specific position
side	BUY or SELL
limitPrice	Required for a limit order
stopPrice	Required for a stop order
tif	GTC, DAY, or GTD; compatibility depends on type
expireDate	Required for GTD

Do not include unrelated or unknown fields.

Do not omit `tif` from the controlled harness. A 3 August 2026 fresh-account BTCUSD MARKET request without `tif` was rejected before execution and created no position or working order. The corrected harness used `tif: "GTC"` for MARKET opens, normal closes, emergency closes, and protective STOPS; subsequent uniquely coded lifecycles passed. Treat GTC as verified deployment evidence and still validate current type/TIF compatibility during discovery.

Verified controlled MARKET-open shape:

```
{
  "orderCode": "<NEW_UNIQUE_ID>",
  "type": "MARKET",
  "instrument": "<DISCOVERED_SYMBOL>",
  "quantity": "<DISCOVERED_VALID_QUANTITY>",
  "positionEffect": "OPEN",
  "side": "<BUY_OR_SELL>",
  "tif": "GTC"
}
```

Safe new-order process

Before POST /accounts/{account}/orders:

1. Re-read account status.
2. Re-read account-specific instrument details.
3. Validate symbol, trading status, quantity, increment, side, order type, price fields, and time in force.
4. Generate a new `orderCode`.
5. Record the intended request without secrets.
6. Require explicit user/strategy authorisation.
7. Send once.
8. Record `orderId` and `updateOrderId`.
9. Reconcile through Push or retrieve open orders/positions.

POST is not idempotent. The unique `orderCode` is the main duplicate-order safeguard.

HTTP 200 and returned internal IDs acknowledge receipt; they do not prove a position opened. Reconcile the returned identifiers against Push, positions, working orders, portfolio, and history where available.

EURUSD requests using the discovered lot values 0.01 and 0.02 returned HTTP 200 and internal IDs but no position. Order history resolved the result as `REJECTED`, filled quantity 0, because the minimum trade size had to equal or exceed 1000.00.

Use:

```
GET /accounts/{encodedAccountCode}/orders/history?with-order-id={orderId}
```

The historical object includes final `status`, executions, filled quantity, and `rejectReason`. This lookup is required before a new uniquely coded request is considered. For the tested EURUSD deployment, reconcile the discovered lot amount with the verified REST base-unit requirement first; do not assume the same mapping for every pair.

On the tested deployment, `GET /orders` exposed some submitted client order codes with the prefix `dxsca-integration-session-code:`. Retain the `orderId/updateOrderId` returned by the mutation and reconcile by internal ID or by a safely normalised client-code suffix. Do not require exact equality with the originally submitted string.

If the client times out after sending, do not immediately send a new order. First search order state using the original identifier. A timeout does not prove that DXtrade rejected the request.

Modifying a working order

Use:

```
PUT /accounts/{encodedAccountCode}/orders
```

Identify the working order with its client or internal order code. Omit `type`; DXtrade states that order type cannot be changed. Live testing confirmed that including `type` produced HTTP 400, error 32.

The portal documents `If-Match/ETag` conditional requests. Velotrade's tested deployment accepted a modification without `If-Match`, but clients should still use the latest ETag where available to reduce race conditions. Do not depend on the server rejecting every stale update.

Only modify an order that is still working. A filled, cancelled, expired, or rejected order is final.

Cancelling a working order

Use:

```
DELETE /accounts/{encodedAccountCode}/orders/{encodedOrderCode}
```

Retrieve the latest order state first. Treat "already final" as a reconciliation outcome, not a reason to create a replacement automatically. During testing, cancelling an already-cancelled order returned HTTP 400, error 1005.

Closing a position

Do not assume that sending an unrelated opposite `OPEN` order safely closes a position.

For a position-based account:

1. Retrieve `GET /accounts/{account}/positions`.
2. Select the exact position.
3. Read `positionCode`, `symbol`, `side`, and `current quantity`.
4. Use the opposite order side.
5. Set `positionEffect` to `CLOSE`.

6. Supply the retrieved `positionCode`.
7. For a full market close, quantity may be omitted; DXtrade uses the current position quantity.
8. Verify the position disappears and no closing order remains working.

For a full position-specific closing STOP or LIMIT, omit quantity as well. Supplying it produced HTTP 400, error 33: Closing STOP/LIMIT orders should not have specified quantity.

Verified protective STOP shape:

```
{
  "orderCode": "<NEW_UNIQUE_ID>",
  "type": "STOP",
  "instrument": "<POSITION_SYMBOL>",
  "positionEffect": "CLOSE",
  "positionCode": "<RETRIEVED_POSITION_CODE>",
  "side": "<OPPOSITE_SIDE>",
  "stopPrice": "<VALID_DISCOVERED_PRICE>",
  "tif": "GTC"
}
```

After placement, retrieve working orders and positively confirm the STOP before considering the position protected. When modifying a working protection with PUT, continue to omit type; for a full closing protection, also omit quantity. Live testing confirmed this modify shape.

Verified full-close shape:

```
{
  "orderCode": "<NEW_UNIQUE_ID>",
  "type": "MARKET",
  "instrument": "<POSITION_SYMBOL>",
  "positionEffect": "CLOSE",
  "positionCode": "<RETRIEVED_POSITION_CODE>",
  "side": "<OPPOSITE_SIDE>",
  "tif": "GTC"
}
```

The tested position code matched the opening order's `orderId`, but retrieve the position rather than relying on that relationship.

Five simultaneous minimum-size positions, each with a separately confirmed closing STOP, were also tested successfully. Before opening another position, include all existing exposure in the margin and loss checks. Protect each new position before proceeding to the next.

Multi-asset live-test matrix

Asset class	Symbol	Minimum quantity	REST execution result	Peak observed margin
Equity	AAOI	0.01	Protected lifecycle passed	USD 0.44

Asset class	Symbol	Minimum quantity	REST execution result	Peak observed margin
Forex	EURUSD	0.01 lot	Rejected; request used 0.01 instead of 1,000 units	USD 0.00
Commodity	XAU	0.01	Protected lifecycle passed	USD 6.72
Index/ETF	SPY	0.01	Protected lifecycle passed	USD 1.24

These 29 July 2026 values came from one fresh challenge account and must be rediscovered. The sequential test finished at zero positions and zero working orders, with USD 0.11 realised loss. The equity representative's official calendar was checked before submission to avoid its mandatory event-close window.

The corrected EURUSD minimum request quantity is 1000 base units. A subsequent protected lifecycle completed successfully: order history showed COMPLETED with all 1,000 units filled, observed margin was USD 22.79, a full-close STOP with quantity omitted was confirmed, the exact position closed, and shutdown was flat. The realised round-trip result was USD -0.26.

A separate fresh-AI regression on another account independently confirmed both opening and closing history as COMPLETED, 1,000 units filled, the same USD 22.79 peak margin, confirmed protection, final flat state, and logout. Its realised result was USD -0.33; the USD 0.07 variation is consistent with different market round trips.

A 3 August 2026 fresh-model acceptance observed peak margins of USD 10.52 (BTCUSD), USD 0.49 (AAOI), USD 23.06 (EURUSD), USD 6.76 (XAU), and USD 1.26 (SPY). These are historical, account- and market-specific evidence, not sizing constants. Rediscover quantities and verify live account metrics every time.

Final-state invariant

After a test or planned strategy shutdown, explicitly verify:

open positions = expected positions
working orders = expected orders

For a controlled connectivity test, both should normally be zero. Closing a program or socket does not cancel orders or close positions automatically.

Rate limits

DXtrade documents standard defaults of:

- login: 1 request per second per IP;

- read operations: 10 requests per second per session;
- trading operations: 10 requests per second per session;
- large-data operations: 1 request per second per session.

These are deployment-configurable and must not be treated as guaranteed Velotrade limits. Honour HTTP 429, back off, and reduce call frequency. Never automatically retry a trading request unless its outcome has first been reconciled.

Canonical mutation reconciliation

An HTTP 200 and returned identifiers acknowledge a request; they do not prove execution.

For every mutation:

1. Generate and retain one unique client `orderCode`.
2. Send the trading POST once.
3. Retain `orderId` and `updateOrderId` from the acknowledgement.
4. Reconcile the same request through working orders, positions, portfolio, Push events, and order history as applicable.
5. Accept that a working `orderCode` may be prefixed with `dxsca-integration-session-code;`; match by internal ID or a safely normalised suffix.
6. If the outcome is unknown, do not generate another order code and retry.
7. For a final history result, normalise only the top-level order record; do not misclassify nested execution or fill objects as orders.

Canonical normalised history result:

```
{
  "orderId": "<REDACTED>",
  "status": "COMPLETED",
  "filledQuantity": "<VALUE>",
  "executions": [],
  "rejectReason": null
}
```

Use the helpers and redacted fixtures described in `RESPONSE_NORMALISATION.md`.

Velotrade DXtrade Push API guide

Status: portfolio, metrics, account-event, account-instrument, quote, candle, unsubscribe, reconnect, and sustained-feed behaviour were verified against an authorised Velotrade challenge accounts through 29 July 2026.

This guide explains how to receive live account and market-data updates. It does not place, modify, or cancel orders.

Why use WebSockets if REST can already trade?

REST and WebSockets solve different problems:

- **REST asks for or changes something now.** Use it to log in, inspect the current state, place an order, modify an order, cancel an order, or request a one-off quote.
- **WebSockets tell you when something changes.** After subscribing, the server pushes order, position, account, and market-data updates without waiting for the client to ask again.

A trading algorithm can place orders using REST alone, but without Push it usually has to poll repeatedly:

Place order → request open orders → wait → request again → wait → request again

With Push, the workflow becomes:

Subscribe once → place order with REST → receive order and portfolio updates

This is useful because it:

- reduces unnecessary REST requests and the chance of hitting rate limits;
- reports fills, rejections, cancellations, and position changes sooner;
- provides live quotes without repeatedly requesting one quote at a time;
- helps keep the algorithm's local state aligned with DXtrade;
- makes it easier to detect disconnections and stale data;
- improves auditability because requests can be correlated with pushed responses and updates.

WebSockets are not a replacement for REST. A resilient integration normally uses both:

1. REST for authentication, discovery, state snapshots, and trading actions.
2. Push for live updates.

3. REST again after reconnecting to obtain a fresh snapshot before trusting newly received updates.

For a very simple script that runs once, reads an account, places one request, and exits, REST may be enough. For a long-running algorithm that needs to know what happened after it sent an order, Push is strongly recommended.

What you connect to

Create two secure WebSocket connections:

Purpose	Verified URI
Account, order, position, instrument, and metrics events	wss://dx.velotrade.com/dxsca-web/?format=JSON
Quotes and candles	wss://dx.velotrade.com/dxsca-web/md?format=JSON

The trailing slash before `?format=JSON` in the business-events URI is required.

`wss://dx.velotrade.com/dxsca-web?format=JSON` and `wss://dx.velotrade.com/dxsca-web/ws?format=JSON` returned HTTP 404 during testing.

The DXtrade portal currently renders a blank value where the WebSocket base URI should appear. Do not copy that blank template and do not invent a socket path.

Before opening either socket

1. Log in through POST `https://dx.velotrade.com/dxsca-web/login`.
2. Use `default` as the login domain.
3. Read `sessionToken` from the successful JSON response.
4. Discover the account through the user/account API.
5. Use the complete account code, such as `default:<account number>`. The number shown in the trading interface is not sufficient by itself.

Never put a username, password, or session token directly in source code, documentation, a Git commit, or an AI prompt. Load secrets from environment variables or a secret manager.

Message envelope

Send JSON objects with:

- `type`: the exact DXtrade request type;
- `requestId`: a unique value generated by the client;
- `timestamp`: the current UTC time in RFC 3339/ISO 8601 form;

- session: the REST login's sessionToken;
- payload: the subscription-specific fields.

Generate a new requestId for every request on a connection. Match replies by comparing the server's inReplyTo value with the request ID.

Clients should ignore unknown fields so additions do not break them.

Business-events example

This request asks for an initial account-portfolio snapshot and subsequent portfolio updates:

```
{
  "type": "AccountPortfoliosSubscriptionRequest",
  "requestId": "portfolio-unique-value",
  "timestamp": "2026-07-24T12:00:00.000Z",
  "session": "<REST_SESSION_TOKEN>",
  "payload": {
    "requestType": "LIST",
    "accounts": ["default:<ACCOUNT_NUMBER>"]
  }
}
```

A successful initial response has type: "AccountPortfolios" and inReplyTo: "portfolio-unique-value".

The same business socket was live-tested successfully with:

- AccountMetricsSubscriptionRequest -> AccountMetrics;
- AccountEventsSubscriptionRequest -> AccountEvents;
- AccountInstrumentDetailsSubscriptionRequest -> AccountInstrumentDetails.

Use a unique request ID for each subscription. Follow the exact payload shape in the DXtrade reference; account lists must contain the full discovered account code.

Market-data example

Discover the account-specific instrument symbol first. For the tested challenge account, Bitcoin was BTCUSD, not BTC/USD.

```
{
  "type": "MarketDataSubscriptionRequest",
  "requestId": "quote-unique-value",
  "timestamp": "2026-07-24T12:00:00.000Z",
  "session": "<REST_SESSION_TOKEN>",
  "payload": {
    "account": "default:<ACCOUNT_NUMBER>",
    "symbols": ["BTCUSD"],
    "eventTypes": [
      {
        "type": "Quote",
        "format": "COMPACT"
      }
    ]
  }
}
```

A successful response has `type: "MarketData"`. Quote updates are contained in `payload.events`.

Read-only quote subscriptions were live-verified for equity AAOI, Forex EURUSD, commodity XAU, and index/ETF SPY. For every representative, the portfolio and quote snapshots correlated, both sockets observed and answered a heartbeat, and both subscriptions returned their matching explicit close response. EURUSD quote delivery succeeded even though a REST MARKET submission did not materialise as a position; market-data permission and trade executability are separate acceptance results.

Candle subscriptions were also live-tested using `candleType: "5m"`, `format: "COMPACT"`, and bounded `fromTime/toTime` values. Quote and candle subscriptions can coexist on the market socket, but each must have its own request ID.

Closing subscriptions

For planned shutdown, send the matching close request with:

- a new unique `requestId`;
- `refRequestId` equal to the original subscription request ID;
- the current session and UTC timestamp.

Live-tested pairs include:

Original request	Close request	Successful response
AccountPortfoliosSubscriptionRequest	AccountPortfoliosCloseSubscriptionRequest	AccountPortfoliosSubscriptionClosed
AccountMetricsSubscriptionRequest	AccountMetricsCloseSubscriptionRequest	AccountMetricsSubscriptionClosed
AccountEventsSubscriptionRequest	AccountEventsCloseSubscriptionRequest	AccountEventsSubscriptionClosed
AccountInstrumentDetailsSubscriptionRequest	AccountInstrumentDetailsCloseSubscriptionRequest	AccountInstrumentDetailsSubscriptionClosed

Original request	Close request	Successful response
MarketDataSubscriptionRequest	MarketDataCloseSubscriptionRequest	MarketDataSubscriptionClosed

Closing an already-final subscription produced Reject error 2, Entity not found. Fresh-account testing also observed a correlated Reject "2" on the first explicit market-data close: the snapshot subscription was already final. Treat numeric 2 and string "2" as the same code. When it correlates to a close request, verify `refRequestId`, record the subscription as final, and do not open a replacement automatically.

Keep the session alive

DXtrade may send:

```
{
  "type": "PingRequest",
  "session": "<REST_SESSION_TOKEN>",
  "timestamp": "2026-07-24T12:00:00.000Z"
}
```

Reply promptly on the same socket:

```
{
  "type": "Ping",
  "session": "<REST_SESSION_TOKEN>",
  "timestamp": "<CURRENT_UTC_TIMESTAMP>"
}
```

Do not confuse this WebSocket Ping message with POST /ping, which renews the REST session.

Heartbeat evidence is timing-dependent. A program that closes immediately after its snapshot may never receive a `PingRequest`, even though its handler is correct. For a heartbeat acceptance test, keep the socket open for a configured observation interval, count received pings, and report separately:

- handler implemented;
- pings observed;
- observation duration.

Reject messages

If DXtrade cannot process a request, it returns type: "Reject". Log:

- `inReplyTo`;
- `payload.errorCode`;
- `payload.description`;
- the outgoing request type and `requestId`;

- the socket purpose, without logging the session token.

Normalise an error code before comparing it:

```
const code = String(message.payload?.errorCode ?? "");
```

The same deployment code may be encoded as a JSON number or string.

The specification describes `inReplyTo` as required, and successful responses in live testing always correlated correctly. One malformed negative business subscription, however, produced Reject error 32 without `inReplyTo`. A diagnostic client must therefore tolerate a missing correlation value on Reject: retain a short bounded record of recent requests by socket, type, and timestamp, but never guess that an uncorrelated Reject belongs to a trading mutation.

Common Push errors include:

Error	Meaning	First checks
1	Authorisation required	Missing, expired, or invalid REST session token
32	Incorrect request parameters	Full account code, symbol, request type, event type, and JSON field names
34	No market-data permission	Confirm the account's Push/market-data entitlement
429	Too many requests	Reduce subscription frequency; do not reconnect in a tight loop
2 or "2"	Subscription missing or already final	For a correlated close Reject, verify the referenced subscription and treat it as final

In live testing, using only the visible account number produced error 32. Using the full default:<account number> code succeeded.

Reconnection rules

- Use exponential back-off with random jitter.
- Do not reconnect continuously or open unnecessary parallel sockets.
- Obtain a new REST token if the old session expired.
- Re-create subscriptions after reconnecting.
- Treat close code 1013 as backpressure: slow down consumption and reduce subscription load before retrying.
- Close both sockets and call `POST /logout` during a planned shutdown.

The final release-candidate test processed more than 20,000 `MarketData` messages, 700 account-metrics updates, and hundreds of Push pings without an unexpected disconnect. A

separate five-minute reconnect soak also succeeded. These counts demonstrate the tested session, not a guaranteed delivery rate.

Compression

Start with `format=JSON` and no compression while developing. DXtrade also documents `compression=gzip`, but application libraries vary in how they negotiate WebSocket compression. Enable it only after the uncompressed flow is working and the chosen library's behaviour has been verified.

Safety and policy

Push subscriptions do not authorise a strategy or activity. Before enabling mutations, complete the live website gate in API safety and live policy check (`API_SAFETY_AND_LIVE_POLICY.md`). If required current official content is unavailable or ambiguous, remain read-only.

Asset-class and catalogue mapping

Velotrade website terminology and the DXtrade API use these mappings:

Website terminology	API assetClass
Stocks	Equity
Indices and ETFs	Index
Commodities	Commodity
Forex	Forex
Crypto	Crypto

Account-specific API discovery controls technical availability for that account. The current Velotrade website controls customer-facing product and policy representations. Counts may differ because of timing, entitlement, account configuration, or catalogue publication.

For read-only reporting, record both counts and label the discrepancy. For a mutation, choose only a symbol clearly supported by both sources. If a symbol or class appears only on one side, remain read-only for it until the operator or Velotrade clarifies the discrepancy.

Never select the first instrument in a class as mutation logic. Read-only examples may choose a representative for discovery evidence, but mutation selection must complete the mandate, policy, event, quantity, price, and margin checks independently.

Response normalisation and redacted fixtures

DXtrade responses may use arrays or named wrappers. Use schema-aware normalisers instead of recursively treating every nested object as the same record type.

The package includes:

- `examples/node/normalizers.mjs`;
- `examples/python/normalizers.py`;
- redacted JSON fixtures under `examples/fixtures/`;
- a Node fixture test run by `npm run fixtures`.

Canonical collections are:

- **metrics**: top-level objects containing account metrics such as `balance`, `equity`, or `margin`;
- **portfolio**: top-level portfolio records containing positions/orders;
- **positions**: records containing `positionCode`, `symbol/instrument`, and `side`;
- **working orders**: order records containing an order identifier and order status/type, excluding nested executions and fills;
- **order history**: top-level order records with status plus nested executions;
- **instruments**: records with `symbol`, `tradingStatus`, and quantity metadata;
- **market data**: event records with `symbol`, `type`, and `quote/candle` fields.

All example output passes through redaction helpers by default. Full payload output requires the explicit `--unsafe-full-output` switch and is local-only; never upload or paste it into an AI conversation.

Troubleshooting Velotrade DXtrade API connections

Start with the exact HTTP status, response content type, DXtrade error code, description, method, and path. Do not diagnose from the HTTP status alone.

Never include a password, session token, `Authorization` header, or unredacted personal data in a support request.

Fast diagnostic sequence

1. Confirm the host is `dx.velotrade.com`.
2. Confirm REST paths begin with `/dxsca-web`.
3. Confirm API login uses `domain: "default"`.
4. Confirm the header is exactly `Authorization: DXAPI <token>`.
5. Call `POST /ping`.
6. Confirm the account was discovered through `/users`.
7. Confirm path values are URL-encoded.
8. Confirm symbols and size rules came from the account-specific instrument endpoint.
9. Read the entire JSON error object.
10. For an uncertain trading result, reconcile order and position state before retrying anything.

Symptom: a successful response fails JSON parsing

Do not assume every successful REST response has a JSON body. The deployed logout endpoint can return HTTP 200 with an empty body.

1. Read the status first.
2. Check content length and content type when present.
3. Read the body as text.
4. Parse JSON only when the body is non-empty and JSON is expected.

An `Unexpected end of JSON input` after logout can therefore be a client parser defect rather than an API rejection.

Symptom: login returns 401, error 3

Likely causes:

- wrong username or password;
- wrong REST domain;
- temporary or permanent lock after failed attempts;
- API permission revoked.

Checks:

- use the trading-account credentials;
- use `default`, not `velotrade`, as the REST domain;
- do not retry in a tight loop;
- verify that the same credentials can access the intended account;
- contact support if the response says the account is locked.

Do not repeatedly test guessed passwords. DXtrade documents a default login limit of one request per second, and repeated failures may trigger a lock.

Symptom: login returns 500, error 110

DXtrade confirmed on 29 July 2026 that this can occur when the underlying authentication service cannot return a clean `SESSION` or `REJECT` result. Known triggers include an expired password or a pending MFA enrolment/challenge. `/dxsca-web/login` maps that additional-step state to the generic HTTP 500, error code 110.

This does not automatically mean the REST service is down, and error 110 does not identify which authentication step is pending.

Action:

1. Stop retrying.
2. Confirm the request used JSON with only `username`, `domain: "default"`, and `password`.
3. Do not add guessed TOTP, security-code, or backup-code fields.
4. Open the Velotrade web login and complete the required password change and MFA enrolment.
5. Retry the documented REST login once.
6. If the error persists, capture the UTC timestamp, complete redacted response, response headers, and source IP for private server-log correlation.

The REST endpoint has no documented programmatic MFA continuation. DXtrade is reviewing the definitive response mapping for expired passwords, required MFA enrolment, required MFA challenge, and incorrect credentials.

Do not propose HMAC as a workaround without operator confirmation. Although HMAC signing is enabled at platform level, customer-specific principals for demo/challenge accounts are not yet confirmed, and the configured service credentials may be affected by the same password/MFA state.

Symptom: login works but another request returns 401

Likely causes:

- missing or malformed `Authorization` header;
- expired token;
- token was logged out or revoked;
- a newly returned token from `POST /ping` was not retained.

Action:

1. Stop trading actions.
2. Check the header construction without logging the token.
3. Authenticate once.
4. Replace the stored token with the latest returned token.
5. Rebuild Push subscriptions if a new session was created.

Symptom: 403

403 means the server understood the request but refused it. Capture the DXtrade error code and description.

Do not treat every 403 as bad credentials. Check:

- user/API permission;
- account status;
- instrument `tradingStatus`;
- order-side or position restrictions;
- current official Velotrade website content and account limits.

Symptom: an account URL returns an HTML 404

First suspect an incorrect path identifier.

Wrong:

```
/accounts/130 000 000/metrics
```

Correct shape:

```
/accounts/default%3A130 000 000/metrics
```

Discover the exact account code from `/users`; do not copy the example number. If the content type is HTML rather than DXtrade JSON, inspect the URL before diagnosing permissions.

Symptom: JSON 404, error 2

DXtrade uses error 2 for inaccessible or missing entities and, in some contexts, unavailable API permission.

Check:

- API surface and base path;
- full account code;
- URL encoding;
- symbol spelling and encoding;
- account ownership and status;
- endpoint permission.

Symptom: 400, error 31

The URI did not match the expected resource shape. Check:

- missing or extra path segments;
- incorrectly encoded path values;
- query parameter names;
- wrong HTTP method.

Symptom: 400, error 32

One or more request parameters are incorrect.

Common verified causes:

- including `type` in a single-order modify request;
- using only the visible account number in a Push subscription.

Also check:

- JSON field names and casing;
- enum spelling;

- symbol;
- event type and format;
- required timestamp/session fields on Push messages;
- negative paging values;
- account-specific size and price rules.

Symptom: 400, error 33

The order request is malformed or semantically incompatible. Examples include a limit order without `limitPrice`, a stop order without `stopPrice`, or incompatible group fields.

Live testing also produced error 33 when `quantity` was supplied on a full position-specific closing STOP. The response stated that closing STOP/LIMIT orders should not specify quantity. Retrieve the position, include its `positionCode`, use `positionEffect`: "CLOSE" and the opposite side, but omit `quantity` for a full closing STOP/LIMIT.

Do not repair the body by trial and error against a trading account. Validate the request locally against the documented order-type matrix first.

Symptom: 404, error 35, Data not ready yet

An unknown symbol in a REST `/marketdata` request produced HTTP 404, error 35, Data not ready yet. Do not interpret that wording as proof of a temporary delay for a valid symbol.

1. Re-run account-specific instrument discovery.
2. Compare the symbol exactly, including punctuation and case.
3. Confirm the instrument is entitled and returned for the account.
4. Retry only after discovery supports the symbol.

Symptom: 409, error 100

A client-originated identifier already exists.

For order placement, the `orderCode` must be unique on the account. Do not change the identifier and resend until you determine whether the original order exists. A duplicate error is evidence that the server has already seen that identifier.

When reconciling, note that `GET /orders` may expose a submitted client code as `dxsca-integration-session-code:<client-code>`. Retain returned internal IDs and compare a safely normalised suffix; exact string equality alone can miss the working order.

Symptom: 400 or 409, error 1005

The referenced order is already closed/final or cannot be cancelled or modified in its current state.

Retrieve current orders and positions. Do not automatically create a new order as a substitute.

Symptom: 409 SERVICE_ERROR or NO_MESSAGE_BUS

First identify which API surface returned it.

The Velotrade website's internal `/api/...` endpoints and the documented DXtrade REST `/dxsca-web/...` endpoints are different interfaces. Live testing observed website 409 errors while DXtrade REST continued to work.

Check:

- exact host, path, and method;
- response content type and body;
- whether `POST /dxsca-web/ping` works;
- whether the error came from website login/session infrastructure or DXtrade REST.

Do not conclude "session collision" or "wrong REST endpoint" from status 409 alone.

Symptom: 412 Precondition Failed

The `If-Match` value does not match current server state.

Retrieve the latest resource and ETag, reconsider the intended modification, then submit a deliberate new request. Do not overwrite current state blindly.

Velotrade testing found that some modify/cancel calls were accepted without a current ETag. Continue using ETags as a client concurrency safeguard; do not assume missing enforcement makes stale writes safe.

Symptom: 429 Too Many Requests

- Stop immediate retries.
- Honour `Retry-After` if supplied.
- Apply exponential back-off with random jitter.
- Reduce polling and use Push for live changes.
- Separate read and trading queues.
- Do not automatically retry a trading request without reconciling it.

Published numeric rate limits are standard DXtrade defaults and may differ on Velotrade.

Symptom: WebSocket handshake returns 404

Use the exact verified URIs:

```
wss://dx.velotrade.com/dxsca-web/?format=JSON
wss://dx.velotrade.com/dxsca-web/md?format=JSON
```

The trailing slash on the business-events URI is required.

These do not work:

```
wss://dx.velotrade.com/dxsca-web?format=JSON
wss://dx.velotrade.com/dxsca-web/ws?format=JSON
```

Symptom: Push returns Reject error 1

The session is missing, invalid, or expired.

- Authenticate through REST.
- Include the exact returned token in each Push message's `session`.
- Reply to `PingRequest`.
- Reauthenticate and resubscribe after expiry.

Symptom: Push returns Reject error 32

Check:

- full account code, including `default`;
- correct socket for the subscription;
- exact request `type`;
- unique `requestId`;
- valid JSON;
- current UTC `timestamp`;
- symbol and event type.

Although `inReplyTo` is documented as required, one live negative request returned error 32 without it. Do not let a missing correlation field crash the receive loop. Attribute an uncorrelated Reject only as a socket-level diagnostic unless other bounded evidence identifies the request.

Symptom: Push returns Reject error 2

The referenced subscription does not exist or is already final. This can occur on a repeated unsubscribe, but fresh-account testing also observed it on the first explicit market-data close because the snapshot subscription was already final.

- normalise the code because it may arrive as numeric 2 or string "2";
- require correlation to the close request and verify `refRequestId`;
- treat the original subscription as final;
- do not create a replacement automatically;
- continue planned socket shutdown when local and REST state are reconciled.

Symptom: instrument discovery has no price tick or precision

Do not invent a tick size. Request an account-authorized quote, retain the decimal places displayed by its bid/ask, and use that only as a conservative candidate precision. Validate it with a safely distant pending LIMIT lifecycle. If the server rejects the price, reconcile the order code, refresh the quote, and obtain instrument guidance rather than blindly retrying.

Symptom: Push returns Reject error 34

Market-data permission is not configured for the user/account. Push access is intended to be enabled for Velotrade challenge accounts, so collect a redacted request and response and escalate if the account should be eligible.

Symptom: WebSocket closes with 1013

DXtrade uses close code 1013 for backpressure when a client does not consume messages quickly enough.

- process messages asynchronously;
- keep the receive loop non-blocking;
- reduce subscriptions;
- avoid expensive work in the socket callback;
- reconnect with back-off;
- obtain a REST snapshot and then recreate subscriptions.

Symptom: order outcome is unknown after a timeout

This is a reconciliation problem, not permission to retry.

1. Keep the original `orderCode`.
2. Query working orders, order history where appropriate, positions, and portfolio.
3. Inspect Push events if the connection was active.
4. Decide whether the original request was accepted.
5. Only then determine whether a new action is needed.

Changing the `orderCode` and resending immediately can create two orders.

Symptom: MARKET POST returns IDs but no position appears

HTTP 200, `orderId`, and `updateOrderId` acknowledge the request but do not prove execution. This was reproduced with EURUSD on two FULL_TRADING test accounts.

1. Keep the returned IDs and original `orderCode`.
2. Query GET `/accounts/{account}/orders/history?with-order-id={orderId}`.
3. Read final `status`, `executions`, `filled quantity`, and `rejectReason`.
4. Check business Push, positions, working orders, portfolio, margin, and balance.
5. If the account is conclusively flat, record `acknowledged-no-position`; do not classify it as a fill.
6. Do not resend with a new code merely to test again.
7. Escalate with redacted account, symbol, UTC time, returned IDs, request shape, and reconciled final state.

The tested EURUSD history resolved the mystery: final status was `REJECTED`, filled quantity was 0, and the reason said the minimum trade size must equal or exceed 1000.00. Discovery expressed the minimum as 0.01 lot, while the order endpoint required base units. On the tested EURUSD deployment:

REST Forex quantity = discovered lots × 100,000

Thus the tested 0.01-lot discovery required `quantity: "1000"`. Treat this as deployment evidence, not a universal contract specification. Confirm current official instrument information and account responses before applying the mapping. Use a fresh unique order code only after a new mutation mandate and normal pre-trade checks.

A corrected 1000-unit EURUSD lifecycle was subsequently live-verified: history status `COMPLETED`, filled quantity 1000, confirmed full-close STOP, USD 22.79 observed margin, exact-position closure, and final flat state.

Symptom: multi-asset margin estimate is implausibly small

Do not apply one $\text{price} \times \text{quantity} \times \text{marginRate}$ formula to every asset class. Forex discovery is lot-based but its REST order quantity is base-unit-based on the tested EURUSD accounts. The verified mapping was 0.01 discovered lot to 1000 REST units; it is not a universal specification.

Use complete contract specifications or a broker margin preview when available. If neither exists, keep testing sequential, use the minimum discovered quantity, read live account margin immediately after execution, and flatten if the authorised ceiling cannot be proven.

Symptom: a new order is rejected after omitting `tif`

Reconcile the original unique `orderCode` and confirm that no position or working order resulted. Do not resend the same identifier. A 3 August 2026 fresh-account BTCUSD request from the packaged harness was rejected before execution when `tif` was absent. The corrected controlled builder adds `tif: "GTC"` to MARKET opens, normal and emergency MARKET closes, and protective STOPS. Validate current order-type/TIF compatibility, then use a new unique code only under the existing mutation mandate.

Information to provide to support

- UTC timestamp and timezone;
- API surface: REST or Push;
- method and path/URI;
- HTTP status or WebSocket close code;
- response content type;
- DXtrade error code and description;
- request type and redacted body;
- whether `/ping` succeeds;
- full account code with the numeric portion partially redacted;
- symbol;
- application version and operating system;
- retry frequency.

Never include credentials or session tokens.

Canonical error reference

Normalise numeric and string error representations before comparison.

Status/code	Verified or documented context	Safe response
401 / 3	Bad credentials/domain or lock state	Use default; stop guessing
500 / 110	Generic non-session/non-reject authentication state	Stop; remediate password/MFA in web UI
404 HTML	Incorrect observed account path shape	Check full URL-encoded account code
404 / 2	Missing/inaccessible entity or permission	Check identifier and access
400 / 31	Malformed URI	Correct method/path/query
400 / 32	Incorrect parameters	Validate fields and account-specific facts
400 / 33	Malformed/incompatible order	Validate order schema; full closing STOP/LIMIT omits quantity
404 / 35	Unknown/not-ready market symbol in test	Repeat account-specific discovery
409 / 100	Duplicate client identifier	Reconcile original; do not retry blindly
1005	Referenced order is final/not cancellable	Retrieve current state
412	ETag mismatch	Refresh state and reconsider
429	Rate limit	Back off; never blindly retry a trade
Push 1	Missing/invalid/expired session	Reauthenticate and resubscribe
Push 2 or "2"	Subscription missing/already final	On correlated close, verify refRequestId and treat as final
Push 32	Incorrect parameters	Check account, symbol, and message fields
Push 34	Market-data permission absent	Confirm entitlement and escalate
WS 1013	Backpressure	Consume faster, reduce load, and back off

A generic 409 does not prove a session collision. Website `/api/...` services and DXtrade `/dxsca-web/...` are different interfaces.

Verification evidence

Historical observations belong here rather than being treated as current constants in operational guides.

29 July 2026 controlled tests

Protected minimum-size REST lifecycles were verified for Equity, Forex, Commodity, Index/ETF, and Crypto representatives. Corrected EURUSD REST quantity was 1,000 base units for a discovered 0.01-lot minimum on two tested accounts. Full position-specific closing STOPs omitted quantity, exact positions closed, and final shutdown was flat.

3 August 2026 fresh-model session

User-supplied redacted session feedback reported five protected asset-class lifecycles with final flat shutdown. Peak observed margins were:

Representative	API class	Peak margin (USD)
BTCUSD	Crypto	10.52
AAOI	Equity	0.49
EURUSD	Forex	23.06
XAU	Commodity	6.76
SPY	Index	1.26

The EURUSD result differed from the earlier USD 22.79 observation, reinforcing that historical margin is evidence, not a constant. Rediscover and verify live metrics on every account and run.

In a later fresh-account BTCUSD acceptance, one preliminary MARKET request was rejected before execution because the packaged harness omitted `tif`; no position or working order resulted. The isolated harness was corrected to use `tif: "GTC"` for MARKET opens and closes. Subsequent uniquely coded protected lifecycles passed. The package now centralises MARKET and STOP construction in tested order builders so the field is not omitted from normal or emergency paths.

API safety and live policy check

Read-only work and the live mutation gate

This package deliberately contains no copy, summary, or interpretation of Velotrade's trading rules. Point-in-time policy text can become stale. Velotrade's official website is the live source of truth.

Read-only authentication, discovery, snapshots, quotes, diagnostics and Push checks may proceed first. Before generating or executing any request that can create, modify, cancel, or close an order or position, complete every step in the canonical mutation gate (MUTATION_GATE.md).

At the beginning of a fresh-account session, before credentials or discovery, ask the operator for the account type or challenge plan, current phase, phase-specific restrictions/objectives, and whether existing activity is expected. Record these as operator-supplied facts and reconfirm type/phase at the mutation gate. This testing workflow does not ask whether the account is test or live.

Step 1 — Retrieve and review the current official pages

Page	URL
Terms and Conditions	https://velotrade.com/terms
Trading Rules	https://velotrade.com/rules
Risk Disclosures	https://velotrade.com/risk-disclosures
API Access	https://velotrade.com/api-access
FAQs	https://velotrade.com/faq
Tradable instruments and current product data	https://velotrade.com/instruments

Step 2 — Follow applicable links

Review every additional account-, plan-, instrument-, market-, or jurisdiction-specific page linked from those sources.

Step 3 — Record the check

Record the URL and UTC retrieval time in the run log. Do not copy the page into this knowledge base or treat a cached result as a permanent rule.

Step 4 — Apply the live pages

Resolve any conflict using the precedence stated on the current official pages. This guide does not restate or interpret that precedence.

Step 5 — Stop when uncertain

If a required page is unavailable, ambiguous, or inconsistent with the intended action, keep mutations disabled and ask Velotrade or the authorised operator for direction. Technical API success is not proof that an action complies with current policy.

Retrieval must respect the website's current Terms and technical access controls. Navigate directly to named pages or use an ordinary interactive browser. Do not crawl, scrape, enumerate or bypass controls. Use the fallback sequence in the mutation gate if a path redirects or is dynamically rendered.

If the API does not expose the account plan, record the authorised operator's statement as operator-supplied evidence. Issuer investor-relations and regulatory pages may confirm factual equity event dates only; they do not replace Velotrade policy or product information. Use the website/API mapping and mismatch procedure in Asset class and catalogue (ASSET_CLASS_AND_CATALOGUE.md).

Credential handling

- Keep usernames and passwords in environment variables or a secret manager.
- Keep `.env` files out of source control.
- Never place credentials or live session tokens in an AI knowledge base.
- Redact `Authorization` headers from logs and screenshots.
- Rotate credentials after accidental disclosure.
- Log out unused REST sessions.
- Do not share one session token between unrelated users or services.

Default-safe software behaviour

A reference integration should:

- start in read-only mode;
- require an explicit configuration switch before enabling mutations;
- validate account and instrument state immediately before an action;
- generate unique order identifiers;
- impose conservative request-rate limits;

- use bounded retries with exponential back-off and jitter;
- never blindly retry an order placement;
- stop trading if state is stale, ambiguous, or disconnected;
- reconcile positions and working orders during startup and shutdown;
- log decisions and identifiers without secrets.

For live positions:

- confirm a position-specific closing STOP before proceeding;
- omit `quantity` from a full closing STOP/LIMIT on the tested deployment;
- reconcile working protection by returned internal ID or the server-prefixed order code;
- protect each position before opening another;
- include aggregate existing exposure in margin and loss checks;
- cancel working protection before a deliberate API close, then verify that neither the position nor a protection order remains.

AI discretion after explicit authorisation

Without explicit delegation, an AI must remain read-only or use mutation inputs selected by the user.

After explicit delegation, an AI may make deterministic test choices of instrument, side, minimum valid quantity, order type, and price. The mandate must identify the authorised account, duration, permitted actions, aggregate margin or exposure ceiling, realised-loss ceiling, protection requirement, and shutdown condition. Every choice must use fresh discovery and quote data, remain within the mandate, and be recorded.

Delegation does not authorise an AI to:

- invent credentials, account codes, symbols, size limits, Push URIs, or policy permissions;
- exceed or reinterpret the user's mandate;
- promise profit or trade merely to chase an aspirational target;
- bypass account restrictions or website access controls;
- assume that a request failed merely because the client timed out;
- expose or retain secrets.

Before any mutation, the AI must state the mandate, affected account, technical safeguards, and completion of the live website gate above. If required live policy facts cannot be verified, it must stop.

Safe negative tests should prefer read-only or guaranteed-rejection cases. Do not deliberately trigger credential lockouts, liquidation, margin calls, rate-limit abuse, or protective-stop fills merely to prove that the server can do so.

Source references

- Velotrade official website (live source of truth): <https://velotrade.com/>
- Velotrade API Access: <https://velotrade.com/api-access>
- Velotrade Rules: <https://velotrade.com/rules>
- Velotrade Terms and Conditions: <https://velotrade.com/terms>
- Velotrade Risk Disclosures: <https://velotrade.com/risk-disclosures>
- Velotrade FAQs: <https://velotrade.com/faq>
- Velotrade instruments: <https://velotrade.com/instruments>
- Velotrade DXtrade developer portal: <https://dx.velotrade.com/developers/#/>
- Official DXtrade demo Push API: <https://demo.dx.trade/developers/#/DXtrade-Push-API>
- Project change register and redacted live-test findings are retained in the controlled workspace.



IMPORTANT DISCLAIMER

THIS GUIDE IS PROVIDED SOLELY TO FACILITATE TECHNICAL IMPLEMENTATION.

It is general informational material and does not constitute legal, regulatory, compliance, tax, financial, investment, risk-management, or trading advice. It does not prescribe a strategy or implementation and does not represent that any example, workflow, control, endpoint, behaviour, or configuration is required, suitable for a particular purpose, complete, continuously available, or free from error.

The trader and implementer are solely responsible for all design decisions, testing, deployment, supervision, credential security, order instructions, trading decisions, risk controls, and compliance with current Velotrade website terms, rules, disclosures, product information, account conditions, and applicable law. Always review the current official Velotrade website before enabling mutations. If this guide conflicts with a current official Velotrade page, the current official page controls.

DXtrade is a third-party platform. Interfaces, responses, permissions, instruments, market conditions, margin, and execution behaviour may change or vary by account. Historical test results and examples do not guarantee future connectivity, execution, performance, profitability, or outcomes.

Nothing in this guide is an offer, solicitation, recommendation, guarantee, or assurance of trading success. Seek independent professional advice where appropriate.