VOCSET API Documentation

Version History

Version	Date	Changes
1.2.5	2026-09-21	Amend and Cancel on `POST /api/trade`. `GET /api/trade` returns `status` and `version`. New `GET /api/trade/{internalTradeID}/revisions`. Client cannot be amended. Cancel needs only `tradeID` + `status`.
1.2.0	2026-08-22	Programmatic API-key rotation (`POST /api/auth/apiKey/rotate`). Company fields accept an ISO 17442 LEI. Executing / full-service brokers may omit `executingBroker` and `clearingBroker`.
1.1.1	2026-02-01	Simplified multi-leg trade format.
1.1.0	2026-01-16	Updated for multi-leg trade support.
1.0.1	2025-12-29	Various small updates.
1.0.0	2024-11-19	Initial release — single-leg trades.

This document describes the **1.2.5** public trade and API-key surface. It is a restricted subset of VOCSET: trade upload, live blotter read, revision history, and credential rotation.

Authentication & Authorisation

All endpoints require an API key. Create a key and secret from your user profile in the VOCSET application.

Send both values on every request:

```
X-API-KEY: <api_key>
X-API-SECRET: <api_secret>
```

Successful authenticated responses include:

```
x-api-key-expiry: 2026-11-20T17:16:12Z
```

That is the ISO-8601 UTC instant when the **current** key expires (60 days from creation). Rotate before then.

An invalid or inactive key is rejected.

Response envelope

Successful and most error bodies look like:

```
{
  "timestamp": "2026-09-21T17:16:12.424011385Z",
  "status": 200,
  "result": {},
  "path": "/api/trade"
}
```

`result` is the payload: an array of per-row outcomes on `POST /api/trade`, an array of trades on `GET /api/trade`, a history object on revisions, or the new key pair on rotation. HTTP `200` on upload does **not** mean every row succeeded — inspect each element's `status`.

Handling trades

- Upload Trades : `POST /api/trade`
- Show Trades : `GET /api/trade`
- Trade Revisions : `GET /api/trade/{internalTradeID}/revisions`

Managing API keys

- Rotate API Key : `POST /api/auth/apiKey/rotate`

Upload Active Trades

Allow the user with a valid API key to upload trades into the daily blotter. The same endpoint creates, amends, and cancels. There is no `PUT` or `PATCH`.

URL : `/api/trade`

Method : `POST`

Auth required : YES (API Key)

Permissions required : None

Request headers

```
X-API-KEY: "<api_key>"
X-API-SECRET: "<api_secret>"
Content-Type: application/json
```

The body is always a **JSON array** of trade objects, including a single-trade upload.

Lifecycle (`status`)

`status`	Meaning	Version
omitted or `New`	Book a new trade	Starts at **1**
`Amend`	Full replacement of a live trade	Increments (**2**, **3**, …)
`Cancel`	Cancel a live trade	Unchanged

Unknown values are rejected. `version` is server-owned and ignored on input. The user `tradeID` is never rewritten.

Uniqueness

A live trade is unique on `(tradeID, tradeDate, executingBroker)`, not on `tradeID` alone. The same `tradeID` may exist for a different date or executing broker.

Duplicates in the same request fail with `Trade ID […] is not unique within file`. A New that collides with a live row fails with `Trade ID […] is not unique`. Historical (EOD-archived) rows are not part of that key: Amend or Cancel after archive fails with "original not available".

Company identifiers

`client`, `executingBroker`, `clearingBroker`, and `carryBroker` accept either:

- the VOCSET **short name** shown in the GUI, or
- an ISO 17442 **LEI** (20 characters, valid checksum).

An invalid checksum returns `Invalid LEI for {field} [{value}]`. A LEI that maps to more than one company returns `Ambiguous LEI for {field} [{value}]`.

Omitting brokers (1.2.0)

Field	Who may omit it	What happens
`executingBroker`	Uploading company is an **executing broker** or **full-service broker**	Filled with the uploader (or the uploader's alias)
`clearingBroker`	Uploading company is an executing or full-service broker, **or** `executingBroker` is supplied	Derived from `clearingAccount`

Everyone else must send both. `clearingAccount` remains mandatory for New and Amend.

Data constraints (New and Amend)

Amend is a **full replacement**: send the same mandatory fields as New, plus `"status": "Amend"`. Optional fields that are omitted or blank on an Amend are stored as **null** (they are not left at the previous value).

```
{
  "tradeID": "<MANDATORY>",
  "status": "New|Amend  <OPTIONAL – omit for New>",
  "traceID": "<OPTIONAL>",
  "tradeDate": "YYYY-MM-DD <MANDATORY> cannot be in the future",
  "side": "Buy|Sell <MANDATORY>",
  "quantity": "676 <MANDATORY> integer – up to 32 digits, no decimals",
  "price": "2009 <MANDATORY> decimal – up to 24 digits + 8 dps",
  "instrumentCode": "String <MANDATORY> contract code only, no maturity",
  "maturity": "YYYY-MM-DD <MANDATORY>",
  "strike": "Decimal <MANDATORY when assetClass=Option>",
  "optionType": "Call|Put <MANDATORY when assetClass=Option>",
  "mic": "String <MANDATORY for Future/Option/Equity>",
  "client": "String <MANDATORY> short name or LEI – immutable on Amend",
  "executingAccount": "String <OPTIONAL>",
  "executingBroker": "String <OPTIONAL for EB/FSB uploaders> short name or LEI",
  "productDescription": "String <OPTIONAL>",
  "clearingAccount": "String <MANDATORY>",
  "clearingBroker": "String <OPTIONAL for EB/FSB, or when executingBroker is sent>",
  "carryBroker": "String <OPTIONAL> only when requested",
  "subAccountCode": "String <OPTIONAL>",
  "comment": "String <OPTIONAL>",
  "executionTime": "DateTime(ISO 8601) <MANDATORY> example: 2024-12-01T14:00:04-05:00",
  "giveupTime": "DateTime(ISO 8601) <OPTIONAL>",
  "assetClass": "Future|Option <MANDATORY>"
}
```

Amend rules

- The original must still be on the **live** blotter (not archived, not cancelled).
- `client` **cannot change.** Cancel the original and re-book if the client is wrong.
- If no business field differs from the live row, the call is rejected (`cannot be amended – no fields changed`). `status` and `version` do not count as a change.
- Confirmation is reset to `Unconfirmed` (including trades that were `AutoConfirmed`).
- Multi-leg: replace the whole `legs` array as a set.

Business fields compared for the no-op check: `side`, `quantity`, `price`, `tradeDate`, `clearingDate`, `maturity`, `currency`, `mic`, `assetClass`, `productDescription`, `instrumentCode`, `instrumentCodeType`, `strike`, `optionType`, `executingBroker`, `executingAccount`, `subAccount`, `clearingBroker`, `clearingAccount`, `carryBroker`, `comment`, `executionTime`, `giveupTime`, `strategyName`, and each leg.

Data constraints (Cancel)

Only `tradeID` and `status` are mandatory. `tradeDate`, `executingBroker`, and `client` are optional disambiguators. **All other fields are ignored.**

```
{
  "tradeID": "<MANDATORY>",
  "status": "Cancel",
  "tradeDate": "YYYY-MM-DD <OPTIONAL – required if more than one live row matches>",
  "executingBroker": "String <OPTIONAL>",
  "client": "String <OPTIONAL>"
}
```

If more than one live row has that `tradeID`, the row is rejected with `matches more than one live trade – include tradeDate`. Cancelling a parent cancels the whole multi-leg structure. Confirmation is left as-is (no new confirm cycle).

Data constraints (Multi-Leg Trades)

Multi-leg trades use a parent-child structure. The parent contains a `legs` array. Strategy **parents** must not send `instrumentCode`, `optionType`, or a product MIC — those live on the legs. The parent exchange is taken from the first resolved leg.

```
{
  "tradeID": "<MANDATORY> unique identifier for parent trade",
  "status": "New|Amend|Cancel <OPTIONAL>",
  "strategyName": "CalendarSpread|CalendarStrip|VerticalSpread|Straddle|Strangle|Butter
  "tradeDate": "YYYY-MM-DD <MANDATORY> cannot be in the future",
  "side": "Buy|Sell <MANDATORY>",
  "quantity": "<MANDATORY>",
  "price": "<MANDATORY> can be 0 for spread trades",
  "instrumentCode": "omit on the parent",
  "maturity": "YYYY-MM-DD <MANDATORY>",
  "client": "String <MANDATORY>",
  "clearingAccount": "String <MANDATORY>",
  "clearingBroker": "String <see Omitting brokers>",
  "executionTime": "DateTime(ISO 8601) <MANDATORY>",
  "assetClass": "Future|Option <MANDATORY>",
  "legs": [
    {
      "tradeID": "<MANDATORY> unique identifier for leg",
      "side": "Buy|Sell <MANDATORY>",
      "quantity": "<MANDATORY>",
      "price": "<MANDATORY>",
      "instrumentCode": "String <MANDATORY>",
      "maturity": "YYYY-MM-DD <MANDATORY>",
      "mic": "String <MANDATORY>",
      "executionTime": "DateTime(ISO 8601) <MANDATORY>",
      "assetClass": "Future|Option <MANDATORY>"
    }
  ]
}
```

Strategy Types and Minimum Leg Requirements

Strategy	Min Legs	Description
CalendarSpread	2	Different expiration dates, same strike
CalendarStrip	2	Buy/Sell consecutive expiries
VerticalSpread	2	Same expiration, different strikes
Straddle	2	Buy/Sell both call and put at same strike
Strangle	2	Buy/Sell call and put at different strikes

Strategy	Min Legs	Description
Butterfly	3	Three strikes with defined wings
Condor	4	Four different strikes
IronButterfly	4	Short butterfly with protective wings
IronCondor	4	Short strangle with protective collars
Strip	3	One call + two puts
Strap	3	Two calls + one put
Custom	2	User-defined structure

Multi-Leg Validation Rules

- Legs inherit `client`, `tradeDate`, `clearingBroker`, `clearingAccount`, `executingBroker`, and `executingAccount` from the parent.
- Multi-leg trades must have at least 2 legs (or more depending on strategy).
- If any leg fails validation, the entire parent trade is rejected.
- One `tradeID` per request element; in-file duplicates still error.

Note for `client`, `executingBroker`, `clearingBroker`, `executingAccount`, `clearingAccount`: use the company codes visible in the VOCSET GUI (or a LEI as above).

Data examples

New (single-leg)

```
[
  {
    "tradeID": "20241119-001",
    "tradeDate": "2024-11-19",
    "side": "Buy",
    "quantity": "676",
    "price": "2009",
    "instrumentCode": "CL",
    "maturity": "2024-12-01",
    "mic": "XNYM",
    "client": "CTCINC",
    "productDescription": "Brent Crude",
    "clearingAccount": "GC123",
    "clearingBroker": "DBAG",
    "executionTime": "2024-11-19T14:00:04-05:00",
    "giveupTime": "2024-11-19T14:01:00-05:00",
    "assetClass": "Future"
  }
]
```

`status` may be omitted; the server treats that as `New`.

Amend (full replacement)

```
[
  {
    "tradeID": "20241119-001",
    "status": "Amend",
    "tradeDate": "2024-11-19",
    "side": "Buy",
    "quantity": "700",
    "price": "2010.5",
    "instrumentCode": "CL",
    "maturity": "2024-12-01",
    "mic": "XNYM",
    "client": "CTCINC",
    "productDescription": "Brent Crude",
    "clearingAccount": "GC123",
    "clearingBroker": "DBAG",
    "executionTime": "2024-11-19T14:00:04-05:00",
    "assetClass": "Future"
  }
]
```

Cancel (identifiers only)

```
[
  {
    "tradeID": "20241119-001",
    "status": "Cancel"
  }
]
```

Disambiguate if needed:

```
[
  {
    "tradeID": "20241119-001",
    "status": "Cancel",
    "tradeDate": "2024-11-19",
    "executingBroker": "JJFUTLTD"
  }
]
```

Multi-Leg Trade Example (Calendar Spread)


```
[
  {
    "tradeID": "ML-20241119-001",
    "strategyName": "CalendarSpread",
    "tradeDate": "2024-11-19",
    "side": "Buy",
    "quantity": "10",
    "price": "0",
    "maturity": "2025-01-01",
    "client": "CTCINC",
    "productDescription": "Crude Oil Calendar Spread",
    "clearingAccount": "GC123",
    "clearingBroker": "DBAG",
    "executionTime": "2024-11-19T14:00:04-05:00",
    "assetClass": "Future",
    "legs": [
      {
        "tradeID": "ML-20241119-001-L1",
        "side": "Buy",
        "quantity": "10",
        "price": "70.50",
        "instrumentCode": "CL",
        "maturity": "2025-01-01",
        "mic": "XNYM",
        "executionTime": "2024-11-19T14:00:04-05:00",
        "assetClass": "Future"
      },
      {
        "tradeID": "ML-20241119-001-L2",
        "side": "Sell",
        "quantity": "10",
        "price": "71.25",
        "instrumentCode": "CL",
        "maturity": "2025-02-01",
        "mic": "XNYM",
        "executionTime": "2024-11-19T14:00:04-05:00",
        "assetClass": "Future"
      }
    ]
  }
]
```

Success Responses

Condition : Payload is well-formed and the user is authenticated. Inspect each element — some rows may still have failed.

Code : `200 OK`


```
{
  "timestamp": "2024-11-21T17:16:12.424011385Z",
  "status": 200,
  "result": [
    {
      "id": "20241119-001",
      "status": "OK",
      "message": ""
    }
  ],
  "path": "/api/trade"
}
```


Success Response (Partial)

Code : `200 OK`

```
{
  "timestamp": "2024-11-21T17:16:12.424011385Z",
  "status": 200,
  "result": [
    {
      "id": "20241119-001",
      "status": "ERROR",
      "message": "Trade ID [20241119-001] is not unique",
      "field": "tradeID",
      "value": "20241119-001"
    },
    {
      "id": "20241119-002",
      "status": "OK",
      "message": ""
    }
  ],
  "path": "/api/trade"
}
```


A mixed batch is possible: HTTP 200 with per-row `status: ERROR`. Only the `OK` rows are booked.

Error table (per-row `message`)

Condition	Typical message
Duplicate `tradeID` in this request	`Trade ID [%s] is not unique within file`
New collides with a live row	`Trade ID [%s] is not unique`
Amend, original archived or missing	`Trade ID [%s] cannot be amended – original not available`
Amend of a cancelled trade	`Trade ID [%s] is cancelled and cannot be amended`
Amend changes `client`	`Trade ID [%s] cannot be amended – client cannot be changed; cancel the original and re-book`

Condition	Typical message
Amend with no business-field change	`Trade ID [%s] cannot be amended – no fields changed`
Cancel, original archived or missing	`Trade ID [%s] cannot be cancelled – original not available`
Cancel matches several live rows	`Trade ID [%s] matches more than one live trade – include tradeDate`
Cancel of an already cancelled trade	`Trade ID [%s] is already cancelled`
Invalid / ambiguous LEI	`Invalid LEI for %s [%s]` / `Ambiguous LEI for %s [%s]`
`executingBroker` required	`%s is required unless the uploading company is an executing or full-service broker`
`clearingBroker` required	`%s is required unless the uploading company is an executing or full-service broker, or executingBroker is supplied`

Error Response

Condition : The payload cannot be parsed, or every row failed in a way that surfaces as a request error.

Code : `400 BAD REQUEST`

Error Response

Condition : Unexpected server error.

Code : `500 INTERNAL SERVER ERROR`

Show Active Trades

Gets the currently **live** blotter for the API key owner's company. Archived (EOD) trades are not included; use Trade Revisions with a known `internalTradeID` if you need history for a specific trade.

URL : `/api/trade`

Method : `GET`

Auth required : YES (API Key)

Permissions required : None

Request headers

```
X-API-KEY: "<api_key>"
X-API-SECRET: "<api_secret>"
```

Null fields are omitted from each trade object.

Success Response

Code : `200 OK`

The array of trades is in `result`.

Details of a `Futures` trade on `IFEU`:

```
{
  "timestamp": "2024-11-21T17:16:12.424011385Z",
  "status": 200,
  "result": [
    {
      "internalTradeID": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "tradeID": "20241119-005",
      "userID": "jj@java2go.com",
      "side": "Sell",
      "quantity": 1,
      "price": 100.00000000,
      "tradeDate": "2024-11-19",
      "currency": "USD",
      "mic": "IFEU",
      "assetClass": "Future",
      "instrumentCode": "B",
      "instrumentCodeType": "Ticker",
      "maturity": "2025-06-01",
      "client": "CTCINC",
      "executingBroker": "JJFUTLTD",
      "clearingBroker": "DBAG",
      "clearingAccount": "GC123",
      "executionTime": "2024-11-19T23:33:00.895319Z",
      "reportedTime": "2024-11-19T23:33:01.492583Z",
      "confirmationStatus": "AutoConfirmed",
      "status": "New",
      "version": 1
    }
  ],
  "path": "/api/trade"
}
```


Details of an `Option` trade on `XNYM` after one amendment:

```
{
  "timestamp": "2024-11-21T17:16:12.424011385Z",
  "status": 200,
  "result": [
    {
      "internalTradeID": "7c9e6679-7425-40de-944b-e07fc1f90ae7",
      "tradeID": "119934867",
      "userID": "jj@java2go.com",
      "side": "Sell",
      "quantity": 200,
      "price": 0.09600000,
      "tradeDate": "2024-10-27",
      "currency": "USD",
      "mic": "XNYM",
      "assetClass": "Option",
      "instrumentCode": "ON",
      "instrumentCodeType": "Ticker",
      "strike": 3.20,
      "optionType": "Call",
      "maturity": "2024-12-01",
      "client": "CTCINC",
      "executingBroker": "JJFUTLTD",
      "clearingBroker": "JPMLLC",
      "clearingAccount": "CTC123",
      "executionTime": "2024-10-27T13:27:33Z",
      "giveupTime": "2024-10-28T13:32:01Z",
      "reportedTime": "2024-10-31T14:54:07.630565Z",
      "confirmationStatus": "Unconfirmed",
      "status": "Amend",
      "version": 2
    }
  ],
  "path": "/api/trade"
}
```


Details of a `CalendarSpread` multi-leg trade on `IFEU`:

```
{
  "timestamp": "2025-01-16T10:00:01.123456Z",
  "status": 200,
  "result": [
    {
      "internalTradeID": "0f14d0ab-9608-4a3f-a08a-33129e6c7a84",
      "tradeID": "ML-20250116-001",
      "userID": "jj@java2go.com",
      "strategyName": "CalendarSpread",
      "side": "Buy",
      "quantity": 10,
      "price": 0.00000000,
      "tradeDate": "2025-01-16",
      "currency": "USD",
      "mic": "IFEU",
      "assetClass": "Future",
      "instrumentCodeType": "Ticker",
      "maturity": "2025-03-01",
      "client": "CTCLTD",
      "executingBroker": "DBAG",
      "clearingBroker": "DBAG",
      "clearingAccount": "GC123",
      "executionTime": "2025-01-16T10:00:00Z",
      "reportedTime": "2025-01-16T10:00:01.123456Z",
      "confirmationStatus": "AutoConfirmed",
      "status": "New",
      "version": 1,
      "legs": [
        {
          "tradeID": "ML-20250116-001-L1",
          "side": "Buy",
          "quantity": 10,
          "price": 76.00000000,
          "mic": "IFEU",
          "assetClass": "Future",
          "instrumentCode": "B",
          "maturity": "2025-03-01",
          "executionTime": "2025-01-16T10:00:00Z",
          "status": "New",
          "version": 1
        },
        {
          "tradeID": "ML-20250116-001-L2",
          "side": "Sell",
```

```
          "quantity": 10,
          "price": 75.50000000,
          "mic": "IFEU",
          "assetClass": "Future",
          "instrumentCode": "B",
          "maturity": "2025-04-01",
          "executionTime": "2025-01-16T10:00:00Z",
          "status": "New",
          "version": 1
        }
      ]
    }
  ],
  "path": "/api/trade"
}
```


Details of a `Butterfly` options multi-leg trade on `IFEU`:

```
{
  "timestamp": "2025-01-16T10:00:01.123456Z",
  "status": 200,
  "result": [
    {
      "internalTradeID": "2c5ea4c0-4067-4a8c-9f0a-1c1d3b4e5f60",
      "tradeID": "ML-20250116-003",
      "userID": "jj@java2go.com",
      "strategyName": "Butterfly",
      "side": "Buy",
      "quantity": 10,
      "price": 0.00000000,
      "tradeDate": "2025-01-16",
      "currency": "USD",
      "mic": "IFEU",
      "assetClass": "Option",
      "instrumentCodeType": "Ticker",
      "strike": 75.00,
      "optionType": "Call",
      "maturity": "2025-06-01",
      "client": "CTCLTD",
      "executingBroker": "DBAG",
      "clearingBroker": "DBAG",
      "clearingAccount": "GC123",
      "executionTime": "2025-01-16T10:00:00Z",
      "reportedTime": "2025-01-16T10:00:01.123456Z",
      "confirmationStatus": "AutoConfirmed",
      "status": "New",
      "version": 1,
      "legs": [
        {
          "tradeID": "ML-20250116-003-L1",
          "side": "Buy",
          "quantity": 10,
          "price": 5.00000000,
          "mic": "IFEU",
          "assetClass": "Option",
          "instrumentCode": "B",
          "maturity": "2025-06-01",
          "strike": 70.00,
          "optionType": "Call",
          "executionTime": "2025-01-16T10:00:00Z"
        },
        {
```

```
          "tradeID": "ML-20250116-003-L2",
          "side": "Sell",
          "quantity": 20,
          "price": 3.00000000,
          "mic": "IFEU",
          "assetClass": "Option",
          "instrumentCode": "B",
          "maturity": "2025-06-01",
          "strike": 75.00,
          "optionType": "Call",
          "executionTime": "2025-01-16T10:00:00Z"
        },
        {
          "tradeID": "ML-20250116-003-L3",
          "side": "Buy",
          "quantity": 10,
          "price": 1.50000000,
          "mic": "IFEU",
          "assetClass": "Option",
          "instrumentCode": "B",
          "maturity": "2025-06-01",
          "strike": 80.00,
          "optionType": "Call",
          "executionTime": "2025-01-16T10:00:00Z"
        }
      ]
    }
  ],
  "path": "/api/trade"
}
```


Fields added in 1.2.5

Field	Meaning
`internalTradeID`	Server UUID for this live (or historical) row. Use it on revisions. Not the user `tradeID`.
`status`	`New` | `Amend` | `Cancel`
`version`	Integer. `1` on New; increments on each Amend; unchanged on Cancel. Read-only.

Field	Meaning
`confirmationStatus`	`Unconfirmed` | `Confirmed` | `Rejected` | `AutoConfirmed`

A cancelled trade remains on the live blotter until end-of-day archive, occupying the uniqueness key until then.

Notes

- The fields `strike` and `optionType` are only returned for `Option` trades.
- Multi-leg trades include a `strategyName` field and a `legs` array.
- Legs inherit `client`, `tradeDate`, `clearingBroker`, `clearingAccount`, `executingBroker`, and `executingAccount` from the parent.
- `tradeID` is always the user-supplied id (no version suffix). Version is its own field.

Error Response

Code : `400 BAD REQUEST` — for example if the key's company cannot be resolved.

Code : `500 INTERNAL SERVER ERROR`

Trade Revisions

Return the current live (or historical) trade plus the snapshots taken each time it was amended or cancelled.

URL : `/api/trade/{internalTradeID}/revisions`

Method : `GET`

Auth required : YES (API Key)

Permissions required : None

`internalTradeID` is the UUID from `GET /api/trade` (`internalTradeID`), **not** the user `tradeID` .

Request headers

```
X-API-KEY: "<api_key>"
X-API-SECRET: "<api_secret>"
```

Success Response

Code : `200 OK`

A New trade with no amendments has `revisions: []` .

```
{
  "timestamp": "2026-09-21T17:16:12.424011385Z",
  "status": 200,
  "result": {
    "current": {
      "internalTradeID": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "tradeID": "20241119-001",
      "side": "Buy",
      "quantity": 700,
      "price": 2010.50000000,
      "tradeDate": "2024-11-19",
      "mic": "XNYM",
      "assetClass": "Future",
      "instrumentCode": "CL",
      "client": "CTCINC",
      "executingBroker": "JJFUTLTD",
      "clearingBroker": "DBAG",
      "clearingAccount": "GC123",
      "confirmationStatus": "Unconfirmed",
      "status": "Amend",
      "version": 2
    },
    "revisions": [
      {
        "revision": 1,
        "version": 1,
        "status": "New",
        "supersededAt": "2026-09-21T16:02:11Z",
        "supersededById": "uploader@broker.com",
        "changedFields": {
          "quantity": { "from": 676, "to": 700 },
          "price": { "from": 2009, "to": 2010.5 },
          "lifecycle_status": { "from": "New", "to": "Amend" }
        }
      }
    ]
  },
  "path": "/api/trade/3fa85f64-5717-4562-b3fc-2c963f66afa6/revisions"
}
```


Field	Meaning
`current`	

Field	Meaning
	The live row (or the historical row after EOD), same shape as `GET /api/trade`.
`revisions[].revision`	The version this snapshot **was** when it was superseded.
`revisions[].version`	Same as `revision` for that snapshot.
`revisions[].status`	Lifecycle of that snapshot (`New` or `Amend` typically).
`revisions[].supersededAt`	When the snapshot was written.
`revisions[].supersededById`	Username of the user who amended or cancelled.
`revisions[].changedFields`	Map of field path → `{ "from": …, "to": … }`. Leg diffs use keys such as `legs[0].price`.

A Cancel writes a snapshot with `lifecycle_status` from the previous status to `Cancel`. Version does not increment.

After EOD archive the same URL still works: `current` is the historical row and revisions are the archived copies.

Error Response

Condition : No live or historical trade with that UUID.

Code : `400 BAD REQUEST`

```
{
  "timestamp": "2026-09-21T17:16:12.424011385Z",
  "status": 400,
  "result": [
    {
      "id": "TRADE_NOT_FOUND",
      "status": "ERROR",
      "message": "Trade [3fa85f64-5717-4562-b3fc-2c963f66afa6] not found"
    }
  ],
  "path": "/api/trade/3fa85f64-5717-4562-b3fc-2c963f66afa6/revisions"
}
```

Condition : The key is not allowed to read that trade.

Code : `403 FORBIDDEN` — `ACTION_UNAUTHORIZED`

Code : `500 INTERNAL SERVER ERROR`

Rotate API Key

Allow a client holding a **valid** API key + secret to programmatically mint a **new** key + secret without logging into the VOCSET application. The current credential is used to authenticate the request; on success the old key/secret is **burned immediately** and the newly minted pair is returned in the response.

This enables headless / server-to-server clients to renew their own credentials before the 60-day expiry, replacing the previous application-only flow.

URL : `/api/auth/apiKey/rotate`

Method : `POST`

Auth required : YES — the *current* API key + secret (presented in the standard headers).

Permissions required : None (the credential authorises rotating **only itself**).

Request headers

```
X-API-KEY: "<current_api_key>"
X-API-SECRET: "<current_api_secret>"
```

Request body : None.

Behaviour

- The presented key + secret are validated. If either is wrong the request is rejected with a generic authentication error (the response does **not** reveal which of the two was invalid).
- A new key and secret are generated using a secure random source:
- `apiKey` — prefixed `vk_` (a non-secret handle)
- `apiSecret` — prefixed `vs_` (128-bit key / 256-bit secret respectively)
- The `apiLabel` is preserved; the key creation date is reset to the time of rotation, restarting the 60-day validity period.
- The old credential stops authenticating **the instant the call commits**.
- A rotation notification email is sent to the key owner, and the rotation is recorded in the audit log (never the secret itself).

Expiry grace period

A key may be rotated up to **7 days past** its 60-day expiry. Beyond this grace window, rotation is refused and the key must be re-issued from the VOCSET application.

Restrictions

- **Inactive users** are refused rotation.

Success Response

Condition : The presented key + secret are valid and within the (expiry + grace) window.

Code : `200 OK`

Headers :

- `Cache-Control: no-store` — the secret must not be cached by intermediaries
- `x-api-key-expiry` — ISO-8601 UTC instant when the **new** key expires (creation + 60 days). The same header is returned on every subsequent authenticated request so clients can rotate ahead of time.

Content example :

```
{
  "timestamp": "2026-07-21T17:16:12.424011385Z",
  "status": 200,
  "result": {
    "apiKey": "vk_8Qw3nR1tZ7yKpL0aB2cD4e",
    "apiSecret": "vs_Nf9Xk2Lp7Qw3nR1tZ7yKpL0aB2cD4eG6hJ8mN0oP2q",
    "apiLabel": "cron-uploader",
    "apiKeyCreationDate": "2026-07-21T17:16:12.424011385Z"
  },
  "path": "/api/auth/apiKey/rotate"
}
```

> **Important** : The response is the **only** copy of the new secret — it is returned once and cannot be retrieved again. Persist it before making any further calls. The old key/secret is invalid immediately, so the client must switch to the new pair for all subsequent requests.

Error Response

Condition : Missing headers, or the presented key/secret is invalid (including a credential that has already been burned by a previous rotation).

Code : `401 UNAUTHORIZED`

Content example :

```
{
  "timestamp": "2026-07-21T17:16:12.424011385Z",
  "status": 401,
  "reason": "AUTHENTICATION_FAILED",
  "message": "Invalid API key or secret",
  "path": "/api/auth/apiKey/rotate"
}
```

Error Response

Condition : The credential belongs to an inactive user.

Code : `403 FORBIDDEN`

Content example :

```
{
  "timestamp": "2026-07-21T17:16:12.424011385Z",
  "status": 403,
  "reason": "ACTION_UNAUTHORIZED",
  "message": "User of API key is inactive",
  "path": "/api/auth/apiKey/rotate"
}
```

Error Response

Condition : The key has no creation date, or is expired beyond the rotation grace window. The key must be re-issued from the VOCSET application.

Code : `400 BAD REQUEST`

Content example :

```
{
  "timestamp": "2026-07-21T17:16:12.424011385Z",
  "status": 400,
  "reason": "API_KEY_EXPIRED",
  "message": "API key expired beyond rotation grace window",
  "path": "/api/auth/apiKey/rotate"
}
```

Error Response

Condition : VOCSET encountered an unexpected / unhandled error while processing.

Code : `500 INTERNAL SERVER ERROR`

Example

```
# Rotate -> 200 with new key/secret/creationDate
curl -sS -X POST https://api.vocset.net/api/auth/apiKey/rotate \
  -H "X-API-KEY: $OLD_KEY" \
  -H "X-API-SECRET: $OLD_SECRET"

# Old creds are now rejected on protected endpoints -> 401
curl -sS -o /dev/null -w "%{http_code}\n" https://api.vocset.net/api/trade \
  -H "X-API-KEY: $OLD_KEY" -H "X-API-SECRET: $OLD_SECRET"

# New creds authenticate -> 200
curl -sS -o /dev/null -w "%{http_code}\n" https://api.vocset.net/api/trade \
  -H "X-API-KEY: $NEW_KEY" -H "X-API-SECRET: $NEW_SECRET"
```

Notes

- Rotation is **self-scoped**: a credential can only rotate itself and the new key inherits the same user, role, and privileges. There is no way to rotate another user's key.
- Concurrent rotations of the same key are **last-write-wins**; the losing caller is left holding a dead secret. Serialise rotations for a given key.
- If a client crashes after the server commits but before it stores the new secret, the old key is already burned and the new secret is lost — recover by re-issuing the key from the VOCSET application.

- Browser clients should not use API-key authentication; API keys are intended for server-to-server use.