



DEVELOPER DOCUMENTATION

API User Guide

A short horizontal bar with a gradient from yellow to purple.

Automate your trading strategies with easy-to-use API calls while limiting traditional infrastructure overhead.

This guide covers the WebSocket API, the platform tools built around it, and everything you need to place your first order.

Contents

Part I Platform Overview

Platform Tour	6
Dashboard	7
Orders	8
API Console	9
AI-Native Documentation	10

Part II Getting Started

Introduction	12
Environments	13
Quickstart	15
Authentication	21
Using the API Console	24
Going live	26

Part III Guides

JSON Data Model	28
Connection lifecycle	30
Order Management	33
LS Connect Adapter	35
Raw WebSocket	38
Place an order	41
Cancel or replace an order	44
Get balances and positions	46
Locates	47

Part IV API Reference

Overview	50
Log on	52
Heartbeat	54
Logout	55
Place an order	56
Place a multi-leg order	61
Cancel an order	66
Replace an order	69
Replace a multi-leg order	74
Request account update	78
Request a locate quote	79
Submit a locate order	81
Execution reports	83
Order status	86
Cancel rejections	88
Positions	90
Account Balance	93
Enums	97
Error Codes	99
Symbology	103
Available Routes	105

Part V Code Examples

Limit	108
Limit On Open	109
Limit On Close	110
Market	111
Market On Open	112
Market On Close	113
Stop	114

Stop-Limit	115
Trailing-Stop	116
Single Leg Call	117
Single Leg Put	118
Credit Spread	119
Debit Spread	120
Cancel Order	121
Cancel Replace Order	122
Account Update	123
Locate Quote Request	124
Locate Order	125

Part VI Tools

Download the adapter	127
Using these docs with AI	128

PART I

Platform Overview

Platform Tour

Dashboard

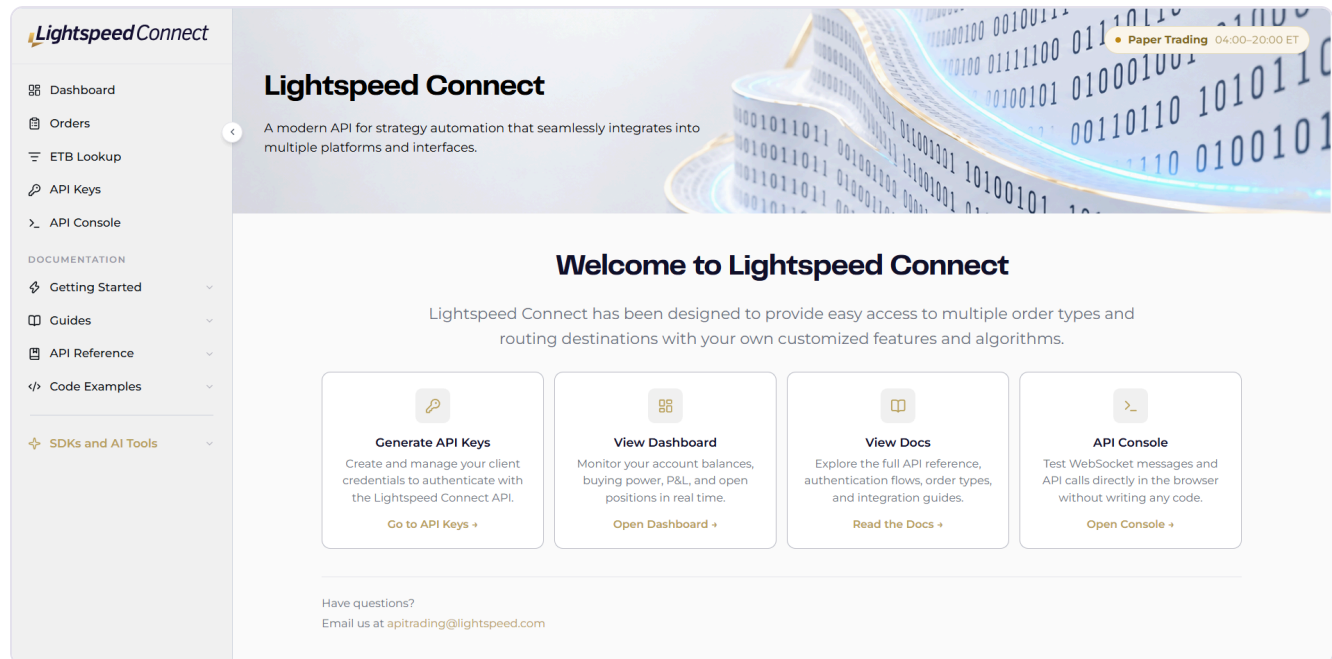
Orders

API Console

AI-Native Documentation

PLATFORM OVERVIEW

Platform Tour



The Lightspeed Connect home screen: quick links to API Keys, Dashboard, Docs, and the API Console.

LS Connect pairs a WebSocket trading API with a full browser platform. Everything documented in this guide works whether you trade through your own code or through the tools built into the platform itself.

What's in the platform

- **Dashboard:** real-time buying power, equity, and P&L, plus every open position.
- **Orders:** a live blotter of every order placed through the account, filterable by status and side.
- **API Console:** send WebSocket messages and read the responses in the browser, no code required.
- **AI-native documentation:** every page on this site is plain markdown at a stable URL, built for coding assistants as well as people.

Where to start

New to LS Connect? [Dashboard](#), [Orders](#), and [API Console](#) give you a tour of the platform first. When you're ready to build, [Part II, Getting Started](#) walks through creating a key and sending your first order.

PLATFORM OVERVIEW

Dashboard

The Dashboard is the account overview for whichever key you're signed in with: available and overnight buying power, current equity, and unrealized, realized, and daily P&L, updated in real time.

The screenshot shows the Lightspeed Connect Dashboard for account lsconnect59. The top section displays six key metrics: Available Buying Power (\$21,484.00), Overnight Buying Power (\$21,484.00), Current Equity (\$100,897.50), Unrealized P&L (\$899.00), Realized P&L (-\$1.50), and Daily P&L (\$897.50). A note states: "Calculations displayed are an estimate for paper trading use only." Below this is a "Positions" table with 2 open positions. The table has columns for Symbol, Account, Qty, Mark Price, Unrealized P&L, Realized P&L, Daily P&L, and Instrument. The positions are for SPY and AMC, both in the lsconnect59 account.

Positions								2 open
SYMBOL	ACCOUNT	QTY	MARK PRICE	UNREALIZED P&L	REALIZED P&L	DAILY P&L	INSTRUMENT	
SPY	lsconnect59	100	763.72	398.00	0.00	398.00	Equity	
AMC	lsconnect59	850	2.52	501.00	-1.50	499.50	Equity	

The Dashboard: buying power, equity, and P&L at the top, open positions listed below.

Below it, the Positions table lists every open position with quantity, price, and P&L broken out the same three ways. It's the fastest way to see where an account stands without writing a line of code.

PLATFORM OVERVIEW

Orders

The Orders page is a live blotter of every order submitted through the account: client and exchange order IDs, status, side, quantity, prices, order type, and time in force.

Orders
View and filter orders submitted through your Lightspeed Connect account.

Paper Trading 04:00-20:00 ET

Q Symbol, account, order id

All Filled 3 Working 2 Cancelled Rejected All sides Buy Sell Short 5 orders Columns

TIME	ACCOUNT	CLIENT ORDER ID	ORDER ID	ORDER STATUS	SYMBOL	SIDE	ORDER QTY	EXECUTED QTY	LE
14:15:30.650	lsconnect59	11f0ff59597d4f5da035c2782a065de7	1260819000068360	Filled	SPY	Sell	10	10	
14:15:23.526	lsconnect59	47f9909f67a342b6903753c515da24e3	1260819000068359	Filled	SPY	Buy	10	10	
14:15:20.708	lsconnect59	baf1666f934745eba5117573487f599f	1260819000068358	Filled	SPY	Buy	10	10	
14:14:06.108	lsconnect59	2688adfa72f24f1cb55f50f3c09591e3	1260819000068349	New	AMC	Buy	100	0	
14:14:03.844	lsconnect59	ac92bb3d17a84122a253dfaabc0d5fd1	1260819000068348	New	AMC	Buy	100	0	

The Orders blotter, filtered to filled orders across all sides.

Filter by status (**All**, **Filled**, **Working**, **Cancelled**, **Rejected**) or side (**Buy**, **Sell**, **Short**), or search by symbol, account, or order ID. It's a fast way to confirm what an integration actually sent, alongside the [ExecutionReport](#) and [OrderSingleStatus](#) messages the API returns for the same activity.

PLATFORM OVERVIEW

API Console

The API Console sends and receives real WebSocket messages from the browser, against whichever key and environment you're signed in with. Pick an order template, adjust the payload, and send it: the acknowledgement, execution reports, and status updates all appear in **Responses**, with the full JSON for any message available in **Details**.

The screenshot displays the LightspeedConnect API Console interface. On the left is a sidebar with navigation links: Dashboard, Orders, ETB Lookup, API Keys, and API Console (selected). Below these are documentation links: Getting Started, Guides, API Reference, Code Examples, and SDKs and AI Tools. The main area is titled 'API Console' and includes a sub-header 'Test WebSocket messages and API calls in the browser.' It features a 'Paper Trading' status bar showing '04:00-20:00 ET' and a 'Connected' indicator. The interface is divided into three main sections: 'Request', 'Responses', and 'Details'. The 'Request' section shows a selected API key 'lsconnect59' and a template for 'OrderSingle' with fields for CLIENTID (LIGHTSPEED) and ACCOUNT (lsconnect59). The 'Responses' section displays a list of 13 messages, including AccountBalanceStatus, PositionStatus, OrderSingleStatus, Logon, PositionUpdate, AccountBalanceUpdate, OrderSingle, OrderSingleUpdate, and AccountBalanceUpdate. The 'Details' section shows the full JSON payload for the selected message, detailing account information, order status, and execution details.

The API Console: a live OrderSingle template sent and filled, with the full response history and detail panel.

It's the fastest way to see the message shapes in this guide in action before writing any code, and a convenient way to keep testing an integration afterward. The full walkthrough is in [Using the API Console](#).

PLATFORM OVERVIEW

AI-Native Documentation

This documentation is written for coding assistants as well as people. Every page is plain markdown at a stable URL: hand any of it to ChatGPT, Claude, Cursor, or your own agent, and it reads the same bytes the site renders, with no scraping and no second copy to drift out of date.

Lightspeed Connect

Using these docs with AI

These docs are written to be read by assistants as well as people. Every page is plain markdown at a stable URL, so you can hand any of it to ChatGPT, Claude, Cursor or your own agent without scraping HTML.

Start here: llms.txt and llms-full.txt

Two machine-readable files cover the entire documentation. **llms.txt** is an index with markdown links to every page, for assistants that discover and retrieve only the pages relevant to a request. **llms-full.txt** is the complete documentation in a single file, for tools that load the full context up front.

Every page is markdown

Take any documentation URL and read the `.md` file behind it. The path is the same:

```
# what you read in the browser
https://lightspeed-connect.connecttrade.com/developers/documentation/guides/adapter

# the same page as markdown
https://lightspeed-connect.connecttrade.com/docs/guides/adapter.md
```

These files need no authentication and are the same bytes the site renders. There is no second copy to drift out of date.

From any page

The rail on the right of every documentation page carries four actions:

ON THIS PAGE

Every page is markdown

From any page

Writing code against the API

Copy page

View as Markdown

Open in ChatGPT

Open in Claude

Every documentation page carries Copy page, View as Markdown, Open in ChatGPT, and Open in Claude actions.

[llms.txt](#) indexes every page for assistants that retrieve only what's relevant to a request. [llms-full.txt](#) is the complete documentation in one file, for tools that load the full context up front. It's what this guide itself was built from.

Every page also carries **Copy page**, **View as Markdown**, **Open in ChatGPT**, and **Open in Claude** actions. See [Using these docs with AI](#) later in this guide for the full detail, including how to read any page's markdown directly from its URL.

PART II

Getting Started

Introduction

Environments

Quickstart

Authentication

Using the API Console

Going live

GETTING STARTED

Introduction

Lightspeed Connect is a WebSocket API for trading equities and single-leg and multi-leg options. The API is organized around JSON messages over a single persistent connection: you send request messages, such as orders, cancels, replaces and locate requests, and receive a real-time stream of updates, such as execution reports, position changes and account balances. It uses WSS (WebSockets over SSL/TLS) and is encrypted by default.

Unlike a REST API, there is no request and response per endpoint: you send a predefined JSON message, and the server pushes updates over the same connection as they happen. Place an order, and its execution reports arrive automatically; you do not poll for them.

What you can trade

Instrument	Orders
Equities	All standard order types
Single-leg options	Calls and puts
Multi-leg options	Strategies of up to four legs

See [Code Examples](#) and [Available Routes](#).

What you can do

- Place, cancel and replace single and multi-leg orders
- Follow orders through execution reports and status changes
- Track positions and account balances
- Request locate quotes and secure borrows on hard-to-borrow symbols

Where to start

1. Pick your endpoint in [Environments](#).
2. Create an API key in [Authentication](#).
3. Send your first order in the [Quickstart](#).

GETTING STARTED

Environments

Lightspeed Connect provides two environments. Paper Trading simulates order execution and is used for development and testing. Production trades live on real markets.

Each environment authenticates you with an [API key](#). You can create an API key in the Lightspeed Connect platform, and you send it when you log on. Keys are not shared between environments: a paper trading key is not valid in production, and a production key is not valid in paper trading.

Endpoints

Environment	Endpoint
Paper Trading	<code>wss://onboarding.connecttrade.com:28052</code>
Production	<code>wss://gateway.connecttrade.com:37382</code>

Moving between environments means changing the endpoint and the API key.

Differences

	Paper Trading	Production
Order execution	Simulated.	Real. Orders are executed on live markets.
Symbols	All supported symbols; execution is simulated.	All supported symbols, subject to market conditions.
Balances and positions	Simulated, for testing account scenarios.	Live account, real funds.
Risk updates	Simulated.	Real-time, on actual positions and balances.

Operating hours

Environment	Hours
Paper Trading	Monday through Friday, 04:00 to 20:00 ET
Production	Monday through Friday, 04:00 to 21:00 ET

Testing in Paper Trading

Any symbol is accepted in paper trading. Orders execute against a simulated market.

GETTING STARTED

Quickstart

Learn how to open a WebSocket connection to Lightspeed Connect, submit an order, and process the stream of updates returned by the server.

Overview

Lightspeed Connect is a WebSocket API for trading equities and equity options. Once connected you send JSON messages and receive real-time updates for:

- Order acknowledgements
- Execution reports as orders fill
- Order status changes, including cancels and rejections
- Position updates
- Account balance updates, including risk figures

For an overview of the API, see [Introduction](#).

Connection URL

Connect to the paper trading WebSocket endpoint at:

```
wss://onboarding.connecttrade.com:28052
```

Production is at:

```
wss://gateway.connecttrade.com:37382
```

Paper Trading simulates order execution. Orders placed there do not reach a live market. It is the environment for development and testing. The example on this page runs against it. For the full comparison, see [Environments](#).

Authentication

Open **API Keys** in the Lightspeed Connect platform and create a paper trading key. The socket opens unauthenticated. The first message you send must be a `Logon` carrying your API key. Keys are per-environment: a paper trading key is not valid in production.

```
{
  "MsgType": "Logon",
  "ApiKey": "YOUR_API_KEY"
}
```

The server replies with a `Logon` carrying your `SessionId`:

```
{
  "MsgType": "Logon",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260",
  "ErrorCode": 0,
  "ErrorText": "Trading OK",
  "SendingTime": "20260825-11:50:48.762"
}
```

NOTE

Keep the `SessionId`. Every message you send from now on must carry it.

See [Authentication](#) for how to generate a key, and the [Log on reference](#) for the full message schema.

Establish the session

The examples on this page use Python 3.8 or later with the `websocket-client` library.

```
pip install websocket-client
```

Open the socket.

```
import websocket

PAPER_ENDPOINT = "wss://onboarding.connecttrade.com:28052"

ws = websocket.WebSocketApp(PAPER_ENDPOINT)
```

When you log on, the server first replays your open orders as status messages, then answers with the `Logon` reply carrying your `SessionId`. Balances and positions follow shortly after. **The `Logon` reply is your go-signal:** once it arrives with `ErrorCode` `0`, you can send orders.

See [Connection lifecycle](#) for the full sequence.

Send an order

A market order to buy 100 shares of `G00G`.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "Symbol": "G00G",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "MARKET",
  "OrderQty": "100",
  "TimeInForce": "DAY",
  "SessionID": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

NOTE

Replace both `ClientID` and `Account` with your own values. In production, the `ClientID` field will be `LSC0`.

TIP: GENERATING CLIENTORDERID

A UUID with the dashes stripped fits the 16 to 32 character range and is unique within the trading day. The complete example below generates one that way.

Quantities and prices are [strings, not numbers](#).

Read what comes back

Three messages arrive, in order:

Message	What it tells you
<code>OrderSingleUpdate</code>	The acknowledgement: your order echoed back with <code>ErrorCode: 0</code> , <code>ErrorText: "Validated OK"</code> and <code>OrderStatus: "PENDING_NEW"</code> .
<code>ExecutionReport</code>	The order executed. Carries <code>LastShares</code> , <code>LastPx</code> and the running <code>CumQty</code> and <code>AvgPrice</code> .
<code>OrderSingleUpdate</code>	The order's current state, here <code>OrderStatus: "FILLED"</code> .

All messages about this order are keyed by your `ClientOrderID`, and you need it to cancel or replace the order.

[Place an order](#) shows each of these payloads in full.

Complete example

A complete example, ready to run against paper trading. Replace `API_KEY`, `ClientID` and `Account` with your own values.

```

import json
import uuid

import websocket

PAPER_ENDPOINT = "wss://onboarding.connecttrade.com:28052"
API_KEY = "YOUR_PAPER_TRADING_API_KEY"

LOGON = {
    "MsgType": "Logon",
    "ApiKey": API_KEY,
}

ORDER = {
    "ClientID": "MyClientID",
    "Account": "MyAccount",
    "MsgType": "OrderSingle",
    "ClientOrderID": uuid.uuid4().hex,
    "Symbol": "GOOG",
    "ExchangeDestination": "NSDQ",
    "Side": "BUY",
    "OrderType": "MARKET",
    "OrderQty": "100",
    "TimeInForce": "DAY",
}

def on_open(ws):
    print("connected")
    print("→ Logon")
    ws.send(json.dumps(LOGON))

def on_message(ws, message):
    msg = json.loads(message)
    print("←", msg.get("MsgType"), msg.get("OrderStatus", ""))

    if msg.get("MsgType") == "Logon":
        # The Logon reply is the go-signal: the open-order snapshot is
        # already behind it, and it carries the SessionId every order needs.
        if msg.get("ErrorCode") != 0:
            print("Logon failed:", msg.get("ErrorText"))
            ws.close()
            return

        ORDER["SessionId"] = msg["SessionId"]
        print("→", ORDER["MsgType"], ORDER["Symbol"])
        ws.send(json.dumps(ORDER))
        return

    if msg.get("ErrorCode"):
        print("error:", msg.get("ErrorText"))

```

Expected output:

```
connected
→ Logon
← OrderSingleStatus (one per open order; none on a fresh account)
← Logon
← AccountBalanceStatus
← PositionStatus
→ OrderSingle G00G
← OrderSingleUpdate PENDING_NEW
← OrderSingleUpdate NEW
← PositionUpdate
← ExecutionReport FILLED
← OrderSingleUpdate FILLED
filled 100 @ 178.23
closed
```

Open orders replay as `OrderSingleStatus` messages before the `Logon` reply. `AccountBalanceStatus` and `PositionStatus` arrive on their own schedule shortly after the logon; their position in the transcript can vary, as can `PositionUpdate` messages between the order updates. The fill price comes from the simulated market.

GETTING STARTED

Authentication

Lightspeed Connect authenticates a session with a **Logon** message. Open the socket, then send a **Logon** carrying your API key as the first message.

Keys are per-environment: a paper trading key is not valid in production, and a production key is not valid in paper trading.

Create an API key

1. Open **API Keys** in the Lightspeed Connect platform.
2. Create a key in the environment you intend to connect to. See [Environments](#) for both endpoints.

The API Keys table masks each key to its last four characters and provides controls to reveal and copy the full value at any time. You do not have to store the key when you create it.

Create production keys the same way once [testing in paper trading is complete](#).

WARNING: KEEP YOUR KEY SECURE

Treat the key as a credential. Do not commit it to source control and do not embed it in client-side code. Supply it to server-side applications from a secrets manager, or from an environment variable where no secrets manager is available.

Log on

Open the socket, then send a **Logon** message. The full message schema is on the [Log on](#) reference page.

```
{
  "MsgType": "Logon",
  "ApiKey": "YOUR_API_KEY"
}
```

A **Logon** comes back carrying the **SessionId** for the session, and **ErrorCode** **0** on success.

```
{
  "MsgType": "Logon",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260",
  "ErrorCode": 0,
  "ErrorText": "Trading OK",
  "SendingTime": "20260825-11:50:48.762"
}
```

NOTE

Keep the `SessionId`. It is required on every message you send after logon, and heartbeats carry it.

NOTE

`ErrorCode: 351` with `ErrorText: "Trading disconnected"` is also a successful logon: the key was accepted and the session is open, but trading is unavailable upstream. Orders are rejected with the same code until it recovers.

A failed logon returns a `Logout` message carrying the error, and the server then closes the socket.

ErrorCode	ErrorText	Cause
118	"REJECT - ApiKey missing"	The <code>Logon</code> carried no <code>ApiKey</code> .
120	"Invalid ApiKey"	The key is not recognized in this environment. A key from the other environment fails the same way.
120	"ApiKey in use already"	The key already has a live connection.

The logon flow in Python:

```
import json
import websocket

PAPER_ENDPOINT = "wss://onboarding.connecttrade.com:28052"
API_KEY = "YOUR_API_KEY_HERE"

LOGON = {
    "MsgType": "Logon",
    "ApiKey": API_KEY,
}

def on_open(ws):
    ws.send(json.dumps(LOGON))

websocket.WebSocketApp(PAPER_ENDPOINT, on_open=on_open).run_forever()
```

The [LS Connect Adapter](#) takes the key as a constructor argument and sends the `Logon` for you.

```
adapter = WS_Adapter(
    connection_str="wss://onboarding.connecttrade.com:28052",
    apiKey="your_api_key_here",
)
```

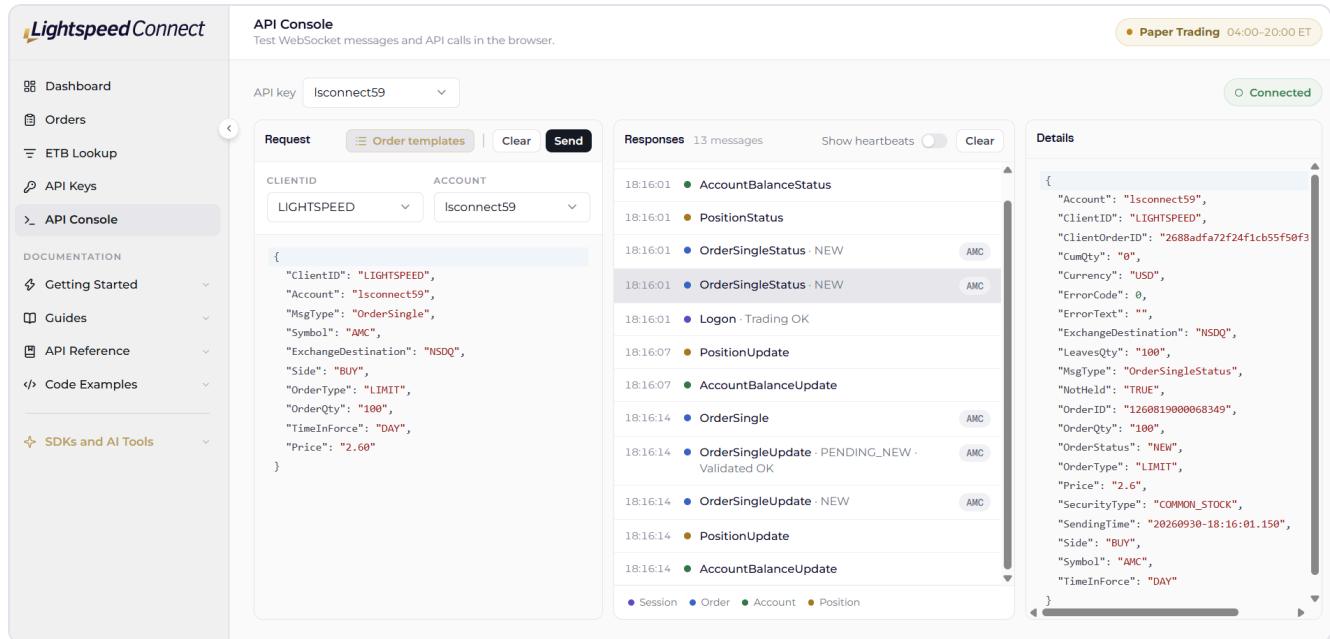
Key lifecycle

- Keys do not expire.
- Each key allows one live connection at a time. A second logon with the same key is rejected with "ApiKey in use already".
- More than one key can be active per account. Give each parallel connection its own key.
- Create and revoke keys in the **API Keys** page of the platform.

GETTING STARTED

Using the API Console

The API Console transmits messages over an authenticated WebSocket session from the browser. No client code is required.



The API Console: Request, Responses, and Details panels.

Connecting

1. Open **API Console** in the Lightspeed Connect platform.
2. Choose a key from the **API key** list, which shows the keys issued for the current environment.
If you have none, [create a key](#) first.

The console opens the session once a key is selected. The status pill beside the list shows its state:

State	Meaning
Connected	The session is open and messages can be sent.
Connecting	The session is being established.
Disconnected	The session is closed, was closed by the server, or failed with an error.

Send is disabled until the state is **Connected**.

Layout

Panel	Contents
Request	The JSON to be transmitted. Order templates populates it, Send transmits it, Clear empties it.
Responses	Replies to messages sent from the Request panel, such as order acknowledgements and cancel rejections, plus the status messages sent when the session opens.
Notifications	Messages the server sends on its own, such as execution reports and position and balance updates. Heartbeats are hidden unless Show heartbeats is enabled.
Details	The full JSON of the selected response or notification.

Send a message

1. Choose a **ClientID**. The **Account** list then narrows to the accounts that ClientID may trade, and the console writes both into the message.
2. Select **Order templates** and choose an order type, or write your message directly in the Request panel. A template fills the panel with a complete order payload.
3. Edit the payload if you need to adjust any values.
4. Select **Send**.

The acknowledgement appears in **Responses** with a generated `ClientOrderID`. Execution reports and position and balance updates follow in **Notifications**. Select a row to see its full payload in **Details**.

NOTE

The console sets `SessionId` and `SendingTime` on every message it sends, overwriting any values entered in the Request panel, and generates a `ClientOrderID` when the payload lacks one.

Examples from the documentation

Every JSON example in this documentation carries a **Try in console** button, which loads the payload into the Request panel.

GETTING STARTED

Going live

Once your integration is tested against [paper trading](#), please notify Lightspeed Support and request production credentials.

Best Practices

- **Review test results:** confirm that every order type your integration uses, along with cancels, replaces, and position and balance updates, behaves as expected in paper trading.
- **Reach out to Lightspeed Support:** raise any open questions about your integration, order routing or risk limits.
- **Obtain production approval:** on successful verification you receive approval to move to the production environment, and production login credentials.

Transitioning to production

- **Replace the endpoint and API key:** point your code at the production endpoint using a production key. See [Environments](#) for both endpoints.

```
import json
import websocket

PROD_ENDPOINT = "wss://gateway.connecttrade.com:37382"
API_KEY = "YOUR_PRODUCTION_API_KEY_HERE"

# The key travels in the Logon message.
LOGON = {"MsgType": "Logon", "ApiKey": API_KEY}

def on_open(ws):
    ws.send(json.dumps(LOGON))

ws = websocket.WebSocketApp(PROD_ENDPOINT, on_open=on_open)
ws.run_forever()
```

- **Conduct production testing:** verify basic connectivity, then place a series of low-risk test trades.
- **Monitor system stability:** watch order executions, real-time updates and system logs under live trading conditions.
- **Begin trading:** once production checks are complete, proceed with live trading.

PART III

Guides

[JSON Data Model](#)

[Connection lifecycle](#)

[Order Management](#)

[LS Connect Adapter](#)

[Raw WebSocket](#)

[Place an order](#)

[Cancel or replace an order](#)

[Get balances and positions](#)

[Locates](#)

GUIDES

JSON Data Model

Every Lightspeed Connect message is a JSON object. `MsgType` identifies the message and the remaining fields carry its payload. Most messages are flat; some carry a nested payload object, such as `account_balances` on `AccountBalanceStatus`.

```
{ "MsgType": "OrderSingle", "Symbol": "GOOG", "Side": "BUY" }
```

Field order and whitespace are not significant. Messages in this documentation are pretty-printed for readability.

Numbers

Decimal values, including prices and quantities, travel as strings rather than JSON numbers, in both directions.

```
{ "Price": "20.52", "OrderQty": "100" }
```

Quoting preserves precision across platforms. A price of `20.52` sent as a floating-point number can be truncated or misread as `20.5`, because languages handle decimals differently. Convert numeric values to strings in your requests, and parse them back with `float()` or `parseFloat()` when you need to calculate.

There are two exceptions:

- `ErrorCode` is a JSON number, not a string.
- The balance and position payloads (`account_balances`, `positions`) carry their values as JSON numbers (`"credit": 1540.12`, `"posAvgPrice": 1.85`): they come from an upstream system and are passed through as-is.

Timestamps

Timestamps are in UTC and use the format `YYYYMMDD-HH:MM:SS.sss`.

```
{ "SendingTime": "20260814-10:38:24.537" }
```

Identifiers

`SessionId` is formatted as a UUID. The `Logon` reply carries it, and it is required on every message sent after logon.

`ClientOrderID` identifies an order. It is required on every order, cancel and replace, must be 16 to 32 characters, and must be unique within the trading day. The shipped clients generate a 32-character id (a UUID v4 without dashes) when your code does not set one.

Example

A limit order, using every convention above.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "GE",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "LIMIT",
  "TimeInForce": "DAY",
  "OrderQty": "100",
  "Price": "20.5"
}
```

GUIDES

Connection lifecycle

A Lightspeed Connect session is one long-lived WebSocket connection. This page describes the messages that cross it and the order they arrive in, independent of the client you use. For how the [LS Connect Adapter](#) surfaces these stages as callbacks, see its guide.

Message sequence

```
sequenceDiagram
    participant Client
    participant Server

    Client->>Server: Open the WebSocket (no credentials)
    Client->>Server: Logon (ApiKey)

    note over Client,Server: Open-order snapshot
    Server->>Client: OrderSingleStatus (each open single order)
    Server->>Client: OrderMultilegStatus (each open multi-leg order)

    Server->>Client: Logon (SessionId, "Trading OK")

    note over Client,Server: Shortly after logon
    Server->>Client: AccountBalanceStatus
    Server->>Client: PositionStatus

    Server->>Client: Live updates, as state changes

    note over Client,Server: Every 60 seconds
    Server->>Client: Heartbeat
    Client-->>Server: Heartbeat (optional)
```

Opening a session

The client opens the socket, which carries no credentials, and sends a `Logon` message holding its API key. The server first replays the account's open orders:

Message	Reports
<code>OrderSingleStatus</code>	Each open single order
<code>OrderMultilegStatus</code>	Each open multi-leg order

The `Logon` reply arrives after them, carrying the `SessionId`. Every message the client sends after logon must carry this `SessionId`, so the `Logon` reply is also the signal that the order

snapshot is complete and orders can be sent.

Balances and positions are not part of this sequence. `AccountBalanceStatus` and `PositionStatus` arrive shortly after the login.

Live updates

Once logged on the server pushes messages as state changes.

Message	Sent when
<code>ExecutionReport</code>	An order executes, in whole or in part
<code>OrderSingleUpdate</code>	An order changes state
<code>OrderCancelReject</code>	A cancel or replace fails
<code>PositionUpdate</code>	A position changes
<code>AccountBalanceUpdate</code>	Balances or risk figures change

Position and balance updates can also arrive between changes; their cadence is not fixed.

Your own requests are acknowledged on the same connection, each with its own update type, such as `OrderCancelUpdate` after a cancel. Each request's reference page describes its acknowledgement.

Heartbeats

The server broadcasts a `Heartbeat` every 60 seconds:

```
{ "MsgType": "Heartbeat", "ErrorCode": 0, "ErrorText": "Session heartbeat", "SendingTime": "20260825-14:02:11.335" }
```

The heartbeat is a keep-alive signal; no reply is needed. The shipped clients echo each heartbeat back, carrying the `SessionId`. The [API Console](#) hides heartbeats behind its **Show heartbeats** toggle.

WARNING

If you do reply to heartbeats, the reply must carry your current `SessionId`. A wrong one fails validation and the server closes the socket.

Reconnection

A dropped connection ends the session. On reconnecting, the sequence starts again from the beginning: the opening snapshot replays, and the client rebuilds its view of open orders and positions before trading resumes. The snapshot carries current state only; messages raised while the connection was down are not stored or resent.

When the engine itself restarts, connected clients receive a `Logout` with `ErrorCode: 299` and `ErrorText: "Session disconnected - Logon required."`, and must log on again.

GUIDES

Order Management

Orders are managed with a small set of messages: place with `OrderSingle` or `OrderMultileg`, change with `OrderCancel`, `OrderReplace` and `OrderReplaceMultileg`, and follow the results through execution reports and order status updates.

New order messages

There are two fundamental message structures used to construct orders. Each message allows the user to:

- Specify the symbol, price, quantity and security type of the order
- Instruct where it should be executed (e.g. using smart order routing, or exchange routing)

The two defined message structures are `OrderSingle` and `OrderMultileg`. Each can be used to place orders for equities and options.

For example, an `OrderSingle` limit order:

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "GE",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "LIMIT",
  "TimeInForce": "DAY",
  "OrderQty": "100",
  "Price": "20.5"
}
```

Order flow

The client sends the order, the server acknowledges it, then reports each change of the order's state. A later `OrderReplace` or `OrderCancel` refers to the order by its `ClientOrderID`, passed as `OrigClientOrderID`.

Each acknowledgement echoes the request with `ErrorMessage: "Validated OK"` and the order's new status. A failed cancel or replace arrives as an `OrderCancelReject`. From then on, `ExecutionReport` and `OrderSingleUpdate` messages report each change of state.

[Place an order](#) and [Cancel or replace an order](#) walk through these flows message by message, and the [Messaging Protocol](#) lists the full message set.

Order states

An order's `OrderStatus` field conveys its state at the time of sending. Fills arrive as `ExecutionReport` messages; other changes of state arrive as `OrderSingleUpdate` messages carrying the current status.

The full value set is on [Enums](#).

In the platform

Orders placed through the API appear in the **Orders** section of the platform. The **Dashboard** shows account balances, risk figures and positions.

GUIDES

LS Connect Adapter

The LS Connect Adapter (`WS_Adapter`) is the Python client shipped with Lightspeed Connect. It logs on, answers heartbeats, reconnects with exponential backoff, and stamps every outgoing message with the fields the API requires. [Download it here](#).

Set up

Requires Python 3.8 or later. Unpack the archive, then create a virtual environment and install the dependencies as described in the package README. The README's quick start runs `console_example.py`, an interactive menu that sends orders, cancels, replaces and locate requests against paper trading.

An asyncio variant, `ws_async_adapter.py`, ships in the same package; see `console_async_example.py` for its usage.

Use it in your code

```

from ws_adapter import WS_Adapter

def on_open(info):
    print(info)

def on_start(msg):
    print("status:", msg["MsgType"])

def on_message(msg):
    print("update:", msg["MsgType"])

def on_error(msg):
    print("error:", msg["ErrorCode"], msg["ErrorText"])

def on_close(msg):
    print("closed:", msg["ErrorText"])

adapter = WS_Adapter(
    connection_str="wss://onboarding.connecttrade.com:28052",
    apiKey="your_api_key_here",
    on_open=on_open,
    on_start=on_start,
    on_message=on_message,
    on_error=on_error,
    on_close=on_close,
)

adapter.start()
adapter.join() # the adapter runs on a background thread

```

The adapter sends the `Logon` itself as soon as the socket opens. Send messages with `adapter.send({...})`. It returns the message as sent on success, and `None` when the message is invalid or the session is not established.

What the adapter adds to each message

Field	Value
<code>SessionId</code>	The id from the <code>Logon</code> reply. Required on every message after logon.
<code>SendingTime</code>	The send time, in UTC.
<code>ClientOrderID</code>	A generated 32-character id, when an order, cancel, replace or locate order does not carry one.
<code>QuoteRequestID</code>	A generated 32-character id, when a <code>LocateRequest</code> does not carry one.

Callbacks

Callback	Called
<code>on_open(info)</code>	Once, when the socket opens.
<code>on_start(msg)</code>	Once per status message: the opening snapshot, and any later <code>*Status</code> message.
<code>on_message(msg)</code>	On the <code>Logon</code> reply and each live update.
<code>on_error(msg)</code>	On a server message with a non-zero <code>ErrorCode</code> , and on socket errors as <code>MsgType</code> <code>"MsgError"</code> with <code>ErrorCode</code> 9999.
<code>on_close(msg)</code>	On a server <code>Logout</code> and when the connection closes.

On disconnection the adapter reconnects on its own, doubling the wait between attempts from 1 second up to a cap of 30 seconds. After a reconnect the opening snapshot replays; see [Connection lifecycle](#).

GUIDES

Raw WebSocket

The API is a plain WebSocket carrying JSON, so any WebSocket library in any language can drive it. The [Quickstart](#) walks through a complete session; this page covers what your client must handle when it does not use the [LS Connect Adapter](#), and gives a minimal skeleton in both synchronous and asyncio Python.

What your client must handle

- Send a `Logon` carrying your API key as the first message, and read the `SessionId` from the reply.
- Include that `SessionId` in every message you send after logon. It is required.
- Optionally reply to server `Heartbeat` messages. A reply is not required, but if you send one it must carry the correct `SessionId`; a wrong one closes the socket.
- Supply your own `ClientOrderID` on orders, cancels and replaces: 16 to 32 characters, unique within the trading day. The shipped clients generate 32-character ids.
- Reconnect after a drop and rebuild state from the replayed opening snapshot. See [Connection lifecycle](#).

Synchronous

Uses the `websocket-client` package, as in the Quickstart.

```

import json
from datetime import datetime, timezone

import websocket

PAPER_ENDPOINT = "wss://onboarding.connecttrade.com:28052"
API_KEY = "YOUR_API_KEY_HERE"

session_id = None

def now():
    return datetime.now(timezone.utc).strftime("%Y%m%d-%H:%M:%S.%F")[:-3]

def on_open(ws):
    ws.send(json.dumps({"MsgType": "Logon", "ApiKey": API_KEY}))

def on_message(ws, raw):
    global session_id
    msg = json.loads(raw)

    if msg.get("MsgType") == "Logon":
        session_id = msg["SessionId"]
    elif msg.get("MsgType") == "Heartbeat":
        ws.send(json.dumps({
            "MsgType": "Heartbeat",
            "SessionId": session_id,
            "SendingTime": now(),
        }))
    else:
        print(msg)

websocket.WebSocketApp(
    PAPER_ENDPOINT,
    on_open=on_open,
    on_message=on_message,
).run_forever()

```

Asyncio

Uses the `websockets` package (`pip install websockets`).

```

import asyncio
import json
from datetime import datetime, timezone

import websockets

PAPER_ENDPOINT = "wss://onboarding.connecttrade.com:28052"
API_KEY = "YOUR_API_KEY_HERE"

def now():
    return datetime.now(timezone.utc).strftime("%Y%m%d-%H:%M:%S.%f")[:-3]

async def main():
    async with websockets.connect(PAPER_ENDPOINT) as ws:
        await ws.send(json.dumps({"MsgType": "Logon", "ApiKey": API_KEY}))
        session_id = None

        async for raw in ws:
            msg = json.loads(raw)

            if msg.get("MsgType") == "Logon":
                session_id = msg["SessionId"]
            elif msg.get("MsgType") == "Heartbeat":
                await ws.send(json.dumps({
                    "MsgType": "Heartbeat",
                    "SessionId": session_id,
                    "SendingTime": now(),
                }))
            else:
                print(msg)

asyncio.run(main())

```

The package also ships an asyncio adapter, `ws_async_adapter.py`, if you would rather not manage this yourself; see [the adapter guide](#).

GUIDES

Place an order

Send an **OrderSingle** and follow it to a fill.

1. Send the order

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "Symbol": "GOOG",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "LIMIT",
  "OrderQty": "100",
  "Price": "20.5",
  "TimeInForce": "DAY",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Set your own `ClientOrderID`: 16 to 32 characters, unique within the trading day. Every later message about the order is keyed by it.

Prices and quantities are [strings, not numbers](#).

2. Read the acknowledgement

The acknowledgement is an **OrderSingleUpdate**. It echoes the order as accepted, including the defaults the server filled in. On acceptance it carries `ErrorCode: 0`, `ErrorText: "Validated OK"` and `OrderStatus: "PENDING_NEW"`.

```
{
  "Account": "MyAccount",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "OrderQty": "100",
  "OrderStatus": "PENDING_NEW",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "GOOG",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "SecurityType": "COMMON_STOCK",
  "NotHeld": "TRUE",
  "ErrorCode": 0,
  "ErrorText": "Validated OK",
  "SendingTime": "20260826-16:15:34.380",
  "MsgType": "OrderSingleUpdate"
}
```

3. Follow the order's life

As the order's state changes you receive [ExecutionReport](#) messages, and [OrderSingleUpdate](#) carrying the current state.

```
{
  "MsgType": "ExecutionReport",
  "Account": "MyAccount",
  "AvgPrice": "20.5",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "CumQty": "100",
  "ExecID": "E-2163476053",
  "ExecTransType": "NEW",
  "LastPx": "20.5",
  "LastShares": "100",
  "OrderID": "O-2163475242",
  "OrderQty": "100",
  "OrderStatus": "FILLED",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "GOOG",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "ExecType": "FILL",
  "LeavesQty": "0",
  "SecurityType": "COMMON_STOCK",
  "RefClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "ErrorCode": 0,
  "SendingTime": "20260826-16:15:34.546"
}
```

GUIDES

Cancel or replace an order

Both operations key off the `ClientOrderID` of the order you want to act on, passed as `OrigClientOrderID`.

Cancel

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderCancel",
  "ClientOrderID": "5c9d1e7f3a2b46c8a0d4f6e8b2c4a6d0",
  "OrigClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41",
  "SessionID": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

WARNING: CANCELS ARE NOT RETROACTIVE

A cancel sent after an order has filled is rejected with an `OrderCancelReject`. Check the ExecutionReport before assuming an order is still working.

A successful cancel is acknowledged by an **OrderCancelUpdate** with `ErrorMessage: "Validated OK"`. The order then reports `PENDING_CANCEL` and, when the cancel completes, `CANCELED`, both as **OrderSingleUpdate** messages. An unsuccessful cancel returns an **OrderCancelReject** with `CancelRejectResponseTo` set to `ORDER_CANCEL`.

Replace

Identify the order with `OrigClientOrderID`. You can change the quantity, the price, the stop price and the time in force; everything else stays as the resting order has it.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderReplace",
  "ClientOrderID": "8a4c6e2d0f9b41d7a3e5c7f9d1b3e5a7",
  "Symbol": "BAC",
  "OrderQty": "200",
  "Price": "20.5",
  "OrigClientOrderID": "7e3a9c5f1b8d4e6a2c4f8b0d6e2a9c53",
  "SessionID": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

A successful replace is acknowledged by an **OrderReplaceUpdate**. A failed replace returns an **OrderCancelReject** with `CancelRejectResponseTo` set to `ORDER_REPLACE`.

GUIDES

Get balances and positions

Ask for the account's current state at any time with **AccountUpdate**.

```
{
  "ClientID": "LSC0",
  "Account": "MyAccount",
  "MsgType": "AccountUpdate",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Two messages come back:

- **AccountBalanceStatus**: buying power, equity, settled and unsettled funds, margin and P&L.
- **PositionStatus**: a `positions` array, one entry per open position.

You do not have to ask for these to receive them. Both also arrive automatically shortly after [login](#), and again as things change: balances as `AccountBalanceUpdate`, positions as `PositionUpdate`.

NOTE

In paper trading, the request can succeed without returning a balance snapshot, because balance and risk data is not always available there.

GUIDES

Locates

Shorting a hard-to-borrow (HTB) symbol takes two messages before the **SELL_SHORT** order: request a quote, then secure the borrow at the quoted rate.

1. Request a locate quote

```
{
  "ClientID": "LSC0",
  "Account": "MyAccount",
  "MsgType": "LocateRequest",
  "Symbol": "ZVZZT",
  "RequestQty": "100",
  "LocateType": "L",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

The quote comes back as a **LocateQuote** carrying the borrow rate, as a price per share. Review it before proceeding, as rates on HTB symbols vary significantly with availability.

2. Secure the borrow

Submit the locate order using the quoted price. **Price** must reflect the rate returned in the quote.

```
{
  "ClientID": "LSC0",
  "Account": "MyAccount",
  "MsgType": "LocateOrder",
  "Symbol": "ZVZZT",
  "RequestQty": "100",
  "LocateType": "L",
  "Price": "0.06613",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

3. Place the short

The confirmation is a **LocateExecution**; a failure comes back as a **LocateOrderReject** with the reason.

Once the locate is confirmed you can send a normal [OrderSingle](#) with `"Side": "SELL_SHORT"`, for up to the located quantity. Locates are valid for the current trading day and expire at end of day.

To check whether a symbol needs a locate, use the **ETB Lookup** page in the Lightspeed Connect platform, which reports a symbol's current borrow classification.

API Reference

Overview

Log on

Heartbeat

Logout

Place an order

Place a multi-leg order

Cancel an order

Replace an order

Replace a multi-leg order

Request account update

Request a locate quote

Submit a locate order

Execution reports

Order status

Cancel rejections

Positions

Account Balance

Enums

Error Codes

Symbology

Available Routes

API REFERENCE

Overview

Lightspeed Connect is a message protocol: every interaction is a JSON object with a `MsgType`, sent or received over one WebSocket session. Formatting conventions live in the [JSON Data Model](#); the order messages arrive in lives in [Connection lifecycle](#).

Each order is referenced by a `ClientOrderID`, unique within a trading day: 16 to 32 characters, supplied by you on every order, cancel and replace. The shipped clients generate one when your code does not set one.

Messages you send

Message	Purpose
<code>Logon</code>	Authenticate the session. The first message on every socket.
<code>Heartbeat</code>	Optionally answer a server heartbeat.
<code>OrderSingle</code>	Place a single order for an equity or equity option.
<code>OrderMultileg</code>	Place a multi-leg options order, up to four legs.
<code>OrderCancel</code>	Cancel an open order. Cancels both <code>OrderSingle</code> and <code>OrderMultileg</code> orders.
<code>OrderReplace</code>	Modify a resting order, within constraints.
<code>OrderReplaceMultileg</code>	Modify a resting multi-leg order.
<code>AccountUpdate</code>	Request current balances and positions on demand.
<code>LocateRequest</code>	Request a borrow-rate quote for a hard-to-borrow symbol.
<code>LocateOrder</code>	Secure the borrow at the quoted rate.

Messages you receive

Message	Sent
<code>Logon</code>	In reply to your Logon, after the open-order snapshot, carrying the <code>SessionId</code> .
<code>Logout</code>	On a failed logon, and broadcast when the engine restarts.
<code>OrderSingleStatus</code> , <code>OrderMultilegStatus</code>	At connect, one per open order, before live updates begin.
<code>PositionStatus</code>	At connect and on request: open positions.
<code>AccountBalanceStatus</code>	At connect and on request: balances and risk figures.
<code>ExecutionReport</code>	On changes in the state of an order.
<code>OrderSingleUpdate</code> , <code>OrderMultilegUpdate</code>	Acknowledges an order and reports each later change of state.
<code>OrderCancelUpdate</code> , <code>OrderReplaceUpdate</code> , <code>OrderReplaceMultilegUpdate</code>	In reply to a cancel or replace , echoing the accepted request.
<code>OrderCancelReject</code>	When a cancel or replace fails, with <code>CancelRejectResponseTo</code> naming the request.
<code>LocateQuote</code> , <code>LocateExecution</code> , <code>LocateRequestReject</code> , <code>LocateOrderReject</code>	In reply to a locate request or locate order .
<code>PositionUpdate</code>	When a position changes; can also arrive between changes.
<code>AccountBalanceUpdate</code>	When balances or risk figures change; can also arrive between changes.
<code>MsgError</code>	When a message cannot be parsed: invalid JSON, a missing <code>MsgType</code> , or a frame over 9,999 bytes.
<code>Heartbeat</code>	Every 60 seconds.

Validation

Validation reports one error at a time: see [Error Codes](#).

API REFERENCE

Log on

Authenticates the session. The first message on every socket; the socket itself opens without credentials. See [Authentication](#) for creating keys and the full walkthrough.

Request

```
{
  "MsgType": "Logon",
  "ApiKey": "YOUR_API_KEY"
}
```

Request Properties

Property	Required	Description
<code>MsgType</code>	Required	Logon
<code>ApiKey</code>	Required	A key for this environment. Each key allows one live connection at a time.

Response

The server first replays your open orders as `OrderSingleStatus` and `OrderMultilegStatus` messages, then replies:

```
{
  "MsgType": "Logon",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260",
  "ErrorCode": 0,
  "ErrorText": "Trading OK",
  "SendingTime": "20260825-11:50:48.762"
}
```

Property	Type	Description
<code>SessionId</code>	string	Required on every message you send for the rest of the session.
<code>ErrorCode</code>	number	<code>0</code> : trading OK. <code>351</code> : logged on, but trading is unavailable upstream; orders reject with the same code until it recovers.
<code>ErrorText</code>	string	"Trading OK", or "Trading disconnected" with code 351.
<code>SendingTime</code>	string	Server timestamp in UTC, <code>YYYYMMDD-HH:MM:SS.sss</code> .

A failed logon is answered with a `Logout` carrying the error, and the server closes the socket. `AccountBalanceStatus` and `PositionStatus` arrive shortly after a successful logon; see [Connection lifecycle](#).

API REFERENCE

Heartbeat

The server broadcasts a `Heartbeat` to every connected socket every 60 seconds.

Message

```
{
  "MsgType": "Heartbeat",
  "ErrorCode": 0,
  "ErrorText": "Session heartbeat",
  "SendingTime": "20260825-14:02:11.335"
}
```

Property	Type	Description
<code>MsgType</code>	string	Heartbeat
<code>ErrorCode</code>	number	<code>0</code>
<code>ErrorText</code>	string	<code>"Session heartbeat"</code>
<code>SendingTime</code>	string	Server timestamp in UTC, <code>YYYYMMDD-HH:MM:SS.sss</code> .

Replying

The heartbeat is a keep-alive signal; no reply is needed. The shipped clients echo each heartbeat back, carrying the `SessionId`:

```
{
  "MsgType": "Heartbeat",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260",
  "SendingTime": "20260825-14:02:11.401"
}
```

WARNING

A reply must carry your current `SessionId`: a wrong one fails validation (ErrorCode 300) and the server closes the socket.

The server answers a client heartbeat only when it carries a `TestReqID`, echoing it back.

API REFERENCE

Logout

Logout is sent by the server when it refuses or ends a session.

Sent when

- **A logon fails.** The `Logout` carries the error, and the server then closes the socket. The failure codes are on [Authentication](#).
- **The engine restarts.** Connected clients receive a broadcast with `ErrorCode: 299` and must log on again.

Message

```
{
  "MsgType": "Logout",
  "ErrorCode": 120,
  "ErrorText": "Invalid ApiKey",
  "SendingTime": "20260825-11:50:48.762"
}
```

Property	Type	Description
<code>MsgType</code>	string	Logout
<code>ErrorCode</code>	number	The reason. See Error Codes .
<code>ErrorText</code>	string	Human-readable reason, for example <code>"Invalid ApiKey"</code> or <code>"Session disconnected - Logon required."</code> .
<code>SendingTime</code>	string	Server timestamp in UTC, <code>YYYYMMDD-HH:MM:SS.sss</code> .

NOTE

To end your session, close the WebSocket. `Logout` is only ever sent by the server: a client-sent `Logout` is rejected with `ErrorCode: 100`.

API REFERENCE

Place an order

Used for sending single orders for equities and options on the Lightspeed Connect platform.

Request

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "Symbol": "GOOG",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "LIMIT",
  "TimeInForce": "DAY",
  "OrderQty": "100",
  "Price": "20.5",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Request Properties

The following table outlines the fields for constructing an OrderSingle message in the Lightspeed Connect API, using JSON field names like "MsgType" that you'll include in your order submissions.

Property	Value	Required	Description
<code>ClientID</code>		Required	In production, the "ClientID" field will be "LSCO".
<code>Account</code>		Required	The server replaces the value with the account bound to your API key.
<code>SessionId</code>		Required	The id from the Logon reply. Required on every message after logon.
<code>MsgType</code>	<code>OrderSingle</code>	Required	
<code>ClientOrderID</code>		Required	Your id for the order: 16 to 32 characters, unique within the trading day. The shipped clients generate one when your code does not set one.
<code>Currency</code>		Optional*	
<code>Symbol</code>		Required	Ticker symbol.
<code>SymbolSfx</code>		Optional	Used for preferred symbols.
<code>Side</code>	<code>BUY</code> <code>SELL</code> <code>SELL_SHORT</code>	Required	
<code>TimeInForce</code>	<code>DAY</code> <code>GOOD_TILL_CANCEL</code> <code>AT_THE_OPENING</code> <code>IMMEDIATE_OR_CANCEL</code>	Optional	System will default to <code>DAY</code> if not specified. Accounts may be restricted to a subset of these values.
<code>OrderQty</code>		Required	Quantity of security ordered.
<code>OrderType</code>	<code>MARKET</code> <code>LIMIT</code> <code>STOP</code> <code>STOP_LIMIT</code> <code>MARKET_ON_CLOSE</code> <code>LIMIT_ON_CLOSE</code>		MARKET - Submit a market order. LIMIT - An order to buy a security at or below a stated price, or to sell a security at or above a stated price. STOP - A stop order to buy which becomes a market order when the security trades at, or above, the stop price. STOP_LIMIT - A stop order to buy which becomes a limit order at the limit price when

MARKET_ON_CLOSE - Indicated price is held to the closing price.

LIMIT_ON_CLOSE - Price is to be around the closing price.

Market-on-open and limit-on-open orders are placed as **MARKET** or **LIMIT** with **TimeInForce: "AT_THE_OPENING"**, and trailing stops as **MARKET** with the trailing instruction; see [Code Examples](#).

Price		Required*	Required depending on the order type specified in OrderType .
StopPrice		Required*	Required for STOP and STOP_LIMIT orders.
ExecInstruction	TRAILING_STOP_PEG	Optional	Set for trailing stop orders.
PegDifference		Required*	The trailing amount; required with ExecInstruction TRAILING_STOP_PEG .
SecurityType	COMMON_STOCK OPTION	Optional	Defaults to COMMON_STOCK . Set OPTION on option orders.
OpenClose	O C	Required*	Required on option orders: O opens a position, C closes one.
MaturityYearMonth		Required*	Required on option orders: expiration year and month, YYYYMM .
MaturityDay		Required*	Required on option orders: expiration day of the month.
PutOrCall	PUT CALL	Required*	Required on option orders.
StrikePrice		Required*	Required on option orders.
SecurityExchange		Optional	We encourage the inclusion of the primary exchange in SecurityExchange to avoid ambiguity in identification of the instrument to be traded.
ExchangeDestination		Optional	See Available Routes for the venue values. Equity orders default to SMART when it is omitted.

**Indicates that these fields are conditionally required based on other criteria.*

Prices and quantities are strings, not numbers.

Response

The order is acknowledged by an **OrderSingleUpdate**. It echoes the order as accepted, including the defaults the server filled in. On acceptance it carries `ErrorCode: 0`, `ErrorText: "Validated OK"` and `OrderStatus: "PENDING_NEW"`.

```
{
  "Account": "MyAccount",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "OrderQty": "100",
  "OrderStatus": "PENDING_NEW",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "GOOG",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "SecurityType": "COMMON_STOCK",
  "NotHeld": "TRUE",
  "ErrorCode": 0,
  "ErrorText": "Validated OK",
  "SendingTime": "20260826-16:15:34.380",
  "MsgType": "OrderSingleUpdate"
}
```

A rejected order comes back the same way, with `OrderStatus: "REJECTED"` and the reason in `ErrorCode` and `ErrorText`:

```
{
  "ClientID": "MyClientID",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "Symbol": "GOOG",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "LIMIT",
  "TimeInForce": "DAY",
  "OrderQty": "100",
  "Price": "20.5",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260",
  "SendingTime": "20260826-16:09:18.790",
  "Account": "MyAccount",
  "TransactTime": "20260826-16:09:18.789",
  "ErrorCode": 200,
  "ErrorText": "##:REJECT - Duplicate ClientOrderID - 9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "OrderStatus": "REJECTED",
  "MsgType": "OrderSingleUpdate"
}
```

Over the order's life you then receive [ExecutionReport](#) messages as its state changes, and [OrderSingleUpdate](#) carrying the current state.

API REFERENCE

Place a multi-leg order

Used for sending multi-leg options orders, supporting up to four legs per order.

Request

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderMultileg",
  "ClientOrderID": "2e6a8c0d4f1b39e5a7c9d1f3b5a7c9e1",
  "Symbol": "WORK",
  "Side": "SELL",
  "Currency": "USD",
  "OpenClose": "C",
  "OrderQty": "4",
  "TimeInForce": "DAY",
  "OrderType": "LIMIT",
  "Price": "-0.24",
  "SecurityType": "MLEG",
  "ExchangeDestination": "SMART",
  "NumLegs": [
    {
      "LegSymbol": "WORK",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "508.00",
      "LegRatioQty": "1",
      "LegSide": "SELL",
      "LegPositionEffect": "CLOSE"
    },
    {
      "LegSymbol": "WORK",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "510.00",
      "LegRatioQty": "1",
      "LegSide": "BUY",
      "LegPositionEffect": "CLOSE"
    }
  ],
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Request Properties

Property	Value	Required	Description
<code>ClientID</code>		Required	In production, the "ClientID" field will be "LSCO".
<code>Account</code>		Required	
<code>SessionId</code>		Required	The id from the Logon reply. Required on every message after logon.
<code>MsgType</code>	<code>OrderMultileg</code>	Required	
<code>ClientOrderID</code>		Required	Your id for the order: 16 to 32 characters, unique within the trading day.
<code>Symbol</code>		Required	The root symbol shared by the legs.
<code>Side</code>	<code>BUY</code> <code>SELL</code>		
<code>Currency</code>			
<code>OpenClose</code>	<code>O</code> <code>C</code>		<code>O</code> opens a position, <code>C</code> closes one.
<code>OrderQty</code>			Number of spreads.
<code>TimeInForce</code>	<code>DAY</code> <code>GOOD_TILL_CANCEL</code> <code>AT_THE_OPENING</code> <code>IMMEDIATE_OR_CANCEL</code>		Defaults to <code>DAY</code> . Accounts may be restricted to a subset of these values.
<code>OrderType</code>	<code>LIMIT</code>	Required	<code>LIMIT</code> in every documented example.
<code>Price</code>			The net price for the spread. Negative for a net credit.
<code>SecurityType</code>	<code>MLEG</code>		Filled by the server when omitted.
<code>ExchangeDestination</code>			
<code>NumLegs</code>		Required	The array of legs.

Leg fields

Property	Value	Required	Description
<code>LegSymbol</code>		Required	The leg's symbol. Defaults to the parent <code>Symbol</code> when omitted.
<code>LegCFIcode</code>	<code>CALL_OPTION</code> <code>PUT_OPTION</code> <code>COMMON_STOCK</code>	Required	
<code>LegSecurityType</code>	<code>OPTION</code> <code>COMMON_STOCK</code>		
<code>LegMaturityDate</code>			Required on option legs.
<code>LegStrikePrice</code>			The leg's strike.
<code>LegRatioQty</code>			The leg's ratio within the spread.
<code>LegSide</code>	<code>BUY</code> <code>SELL</code>		
<code>LegPositionEffect</code>	<code>OPEN</code> <code>CLOSE</code>		

Response

The order is acknowledged by an **OrderMultilegUpdate** echoing it with `ErrorCode: 0`, `ErrorText: "Validated OK"` and `OrderStatus: "PENDING_NEW"`. The echo carries the legs, each with a server-assigned `LegRefID` and its own `OrderStatus`, and replies spell the security type `MULTI_LEG_INSTRUMENT`.

```

{
  "Account": "MyAccount",
  "ClientOrderID": "2e6a8c0d4f1b39e5a7c9d1f3b5a7c9e1",
  "Currency": "USD",
  "OrderQty": "4",
  "OrderType": "LIMIT",
  "Price": "0.24",
  "Rule80A": "AGENCY_SINGLE_ORDER",
  "Side": "SELL",
  "Symbol": "WORK",
  "TimeInForce": "DAY",
  "OpenClose": "CLOSE",
  "ExchangeDestination": "SMART",
  "ClientID": "MyClientID",
  "SecurityType": "MULTI_LEG_INSTRUMENT",
  "NumLegs": [
    {
      "LegSymbol": "WORK",
      "OrderStatus": "PENDING_NEW",
      "LegPositionEffect": "CLOSE",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "510.00",
      "LegRatioQty": "1",
      "LegSide": "BUY",
      "LegRefID": "2-365297669"
    },
    {
      "LegSymbol": "WORK",
      "OrderStatus": "PENDING_NEW",
      "LegPositionEffect": "CLOSE",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "508.00",
      "LegRatioQty": "1",
      "LegSide": "SELL",
      "LegRefID": "1-365297668"
    }
  ],
  "ErrorCode": 0,
  "ErrorText": "Validated OK",
  "SendingTime": "20260826-16:37:10.621",
  "OrderStatus": "PENDING_NEW",
  "MsgType": "OrderMultiLegUpdate"
}

```

Over the order's life you then receive per-leg **ExecutionReport** messages, marked

`MultiLegReportingType: "INDIVIDUAL_LEG_OF_A_MULTI_LEG_SECURITY"` with the leg's `LegRefID` and

option identifiers, and further **OrderMultilegUpdate** messages carrying each leg's progress.

`OrderMultilegStatus` appears only in the [connect snapshot](#).

API REFERENCE

Cancel an order

Used to cancel open orders. A successful cancel is acknowledged by an `OrderCancelUpdate` message, while an unsuccessful cancel results in an `OrderCancelReject` message with the `CancelRejectResponseTo` tag set to `ORDER_CANCEL`. Note that both `OrderSingle` and `OrderMultileg` orders are canceled with this message.

Request

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderCancel",
  "ClientOrderID": "5c9d1e7f3a2b46c8a0d4f6e8b2c4a6d0",
  "OrigClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Request Properties

Property	Value	Required	Description
ClientID		Required	In production, the "ClientID" field will be "LSCO".
Account		Required	
SessionId		Required	The id from the Logon reply. Required on every message after logon.
MsgType	OrderCancel	Required	
ClientOrderID		Required	A new id for the cancel request itself: 16 to 32 characters, unique within the trading day.
OrigClientOrderID		Required	Refers to the ClientOrderID of the order you wish to cancel.

Responses

OrderCancelUpdate

A successful cancel is acknowledged by an **OrderCancelUpdate** echoing the request with `ErrorCode: 0` and `ErrorText: "Validated OK"`. The echo includes the `Symbol`, `Side` and

`OrderQty` of the resting order, which the server fills in for you, and `OrderStatus` carries the order's state at the time of the acknowledgement.

```
{
  "Account": "MyAccount",
  "ClientOrderID": "5c9d1e7f3a2b46c8a0d4f6e8b2c4a6d0",
  "OrderQty": "100",
  "OrigClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41",
  "Side": "BUY",
  "Symbol": "ORCL",
  "ClientID": "MyClientID",
  "SecurityType": "COMMON_STOCK",
  "OptOrLS": "LSConnect",
  "ErrorCode": 0,
  "ErrorText": "Validated OK",
  "SendingTime": "20260826-15:21:11.534",
  "OrderStatus": "NEW",
  "MsgType": "OrderCancelUpdate"
}
```

OrderSingleUpdate

The order's progress then arrives as `OrderSingleUpdate` messages: first with `OrderStatus: "PENDING_CANCEL"`, then, when the cancel completes, with the final `OrderStatus: "CANCELED"`.

```
{
  "Account": "MyAccount",
  "ClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41",
  "CumQty": "0",
  "OrderID": "0-2163475118",
  "OrderQty": "100",
  "OrderStatus": "PENDING_CANCEL",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "ORCL",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "LeavesQty": "100",
  "SecurityType": "COMMON_STOCK",
  "NotHeld": "TRUE",
  "ErrorCode": 0,
  "ErrorText": "",
  "SendingTime": "20260826-15:21:11.540",
  "MsgType": "OrderSingleUpdate"
}
```

```
{
  "Account": "MyAccount",
  "ClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41",
  "CumQty": "0",
  "OrderID": "0-2163475118",
  "OrderQty": "100",
  "OrderStatus": "CANCELED",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "ORCL",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "LeavesQty": "0",
  "SecurityType": "COMMON_STOCK",
  "NotHeld": "TRUE",
  "ErrorCode": 0,
  "ErrorText": "",
  "SendingTime": "20260826-15:21:11.556",
  "MsgType": "OrderSingleUpdate"
}
```

OrderCancelReject

An unsuccessful cancel returns an **OrderCancelReject** with **CancelRejectResponseTo** set to **ORDER_CANCEL**, and **OrderStatus** carrying the order's current state. This example shows a cancel rejected because the order filled first:

```
{
  "MsgType": "OrderCancelReject",
  "ClientOrderID": "5c9d1e7f3a2b46c8a0d4f6e8b2c4a6d0",
  "OrderID": "0-2163475290",
  "OrderStatus": "NEW",
  "OrigClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41",
  "ErrorText": "Cancel request rejected during T5",
  "CancelRejectReason": "BROKER_OPTION",
  "ClientID": "MyClientID",
  "CancelRejectResponseTo": "ORDER_CANCEL",
  "ErrorCode": 1300,
  "SendingTime": "20260826-16:30:37.933",
  "RefClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41"
}
```

API REFERENCE

Replace an order

Modifies existing orders submitted through OrderSingle or OrderMultileg, within allowed constraints. Attempting to modify an order with invalid attributes triggers an OrderCancelReject message with CancelRejectResponseTo set to ORDER_REPLACE.

Request

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderReplace",
  "ClientOrderID": "8a4c6e2d0f9b41d7a3e5c7f9d1b3e5a7",
  "Symbol": "BAC",
  "OrderQty": "200",
  "Price": "20.5",
  "OrigClientOrderID": "7e3a9c5f1b8d4e6a2c4f8b0d6e2a9c53",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Request Properties

Property	Value	Required	Description
<code>ClientID</code>		Required	In production, the "ClientID" field will be "LSCO".
<code>Account</code>		Required	
<code>SessionId</code>		Required	The id from the Logon reply. Required on every message after logon.
<code>MsgType</code>	<code>OrderReplace</code>	Required	
<code>ClientOrderID</code>		Required	A new id for the replacing order: 16 to 32 characters, unique within the trading day.
<code>Symbol</code>			Optional; the server always uses the resting order's symbol.
<code>OrderQty</code>		Required	The new quantity.
<code>Price</code>		Required*	The new price. Price or StopPrice must be present.
<code>OrigClientOrderID</code>		Required	The ClientOrderID on the order you want to replace.

The alterable attributes are `OrderQty`, `Price`, `StopPrice` and `TimeInForce`. `Symbol`, `Side` and the security type always come from the resting order, and `OrderType` may change only from LIMIT to MARKET; any other type change is rejected.

Responses

OrderReplaceUpdate

A successful replace is acknowledged by an **OrderReplaceUpdate** echoing the request with `ErrorCode: 0` and `ErrorText: "Validated OK"`. `OrderStatus` carries the order's state at the time of the acknowledgement.

```
{
  "Account": "MyAccount",
  "ClientOrderID": "8a4c6e2d0f9b41d7a3e5c7f9d1b3e5a7",
  "OrderQty": "200",
  "OrderType": "LIMIT",
  "OrigClientOrderID": "7e3a9c5f1b8d4e6a2c4f8b0d6e2a9c53",
  "Price": "20.5",
  "Rule80A": "AGENCY_SINGLE_ORDER",
  "Side": "BUY",
  "Symbol": "BAC",
  "TimeInForce": "DAY",
  "ClientID": "MyClientID",
  "SecurityType": "COMMON_STOCK",
  "OptOrLS": "LSConnect",
  "ErrorCode": 0,
  "ErrorText": "Validated OK",
  "SendingTime": "20260826-16:23:13.719",
  "OrderStatus": "NEW",
  "MsgType": "OrderReplaceUpdate"
}
```

OrderSingleUpdate

The order's progress then arrives as **OrderSingleUpdate** messages: first with **OrderStatus:** **"PENDING_REPLACE"**, then, when the replace completes, with **OrderStatus:** **"REPLACED"** carrying the new quantity. These updates carry the ORIGINAL order's **ClientOrderID**, while **ExecutionReport** messages after the replace carry the replacing **ClientOrderID**, with **OrigClientOrderID** and **RefClientOrderID** pointing at the original.

```
{
  "Account": "MyAccount",
  "ClientOrderID": "7e3a9c5f1b8d4e6a2c4f8b0d6e2a9c53",
  "CumQty": "0",
  "OrderID": "0-2163475267",
  "OrderQty": "100",
  "OrderStatus": "PENDING_REPLACE",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "BAC",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "LeavesQty": "100",
  "SecurityType": "COMMON_STOCK",
  "NotHeld": "TRUE",
  "ErrorCode": 0,
  "ErrorText": "",
  "SendingTime": "20260826-16:23:13.726",
  "MsgType": "OrderSingleUpdate"
}
```

```
{
  "Account": "MyAccount",
  "ClientOrderID": "7e3a9c5f1b8d4e6a2c4f8b0d6e2a9c53",
  "CumQty": "0",
  "OrderID": "0-2163475267",
  "OrderQty": "200",
  "OrderStatus": "REPLACED",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "BAC",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "LeavesQty": "200",
  "SecurityType": "COMMON_STOCK",
  "NotHeld": "TRUE",
  "ErrorCode": 0,
  "ErrorText": "",
  "SendingTime": "20260826-16:23:13.732",
  "MsgType": "OrderSingleUpdate"
}
```

OrderCancelReject

Failure to update the order returns an **OrderCancelReject** with **CancelRejectResponseTo** set to **ORDER_REPLACE**, and **OrderStatus** carrying the order's current state:

```
{
  "MsgType": "OrderCancelReject",
  "ClientOrderID": "8a4c6e2d0f9b41d7a3e5c7f9d1b3e5a7",
  "OrderID": "0-2163475267",
  "OrderStatus": "NEW",
  "OrigClientOrderID": "7e3a9c5f1b8d4e6a2c4f8b0d6e2a9c53",
  "ErrorText": "Replace request rejected",
  "CancelRejectReason": "BROKER_OPTION",
  "ClientID": "MyClientID",
  "CancelRejectResponseTo": "ORDER_REPLACE",
  "ErrorCode": 1300,
  "SendingTime": "20260826-16:23:14.101",
  "RefClientOrderID": "7e3a9c5f1b8d4e6a2c4f8b0d6e2a9c53"
}
```

API REFERENCE

Replace a multi-leg order

Modifies multi-leg orders initially placed using OrderMultileg, with similar constraints and handling.

Request

```
{
  "ClientID": "MyClientID",
  "MsgType": "OrderReplaceMultileg",
  "ClientOrderID": "6f0b2d4a8c3e51f7b9d1e3a5c7f9b1d3",
  "OrderQty": "8",
  "Price": "0.24",
  "OrigClientOrderID": "b2d7e9f4a1c6483e8d0f2a6c4e8b1d75",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Do not resend the legs: the server reconstructs them from the resting order, and the acknowledgement echoes them back with their original `LegRefID` values.

Request Properties

Property	Value	Required	Description
<code>ClientID</code>		Required	In production, the "ClientID" field will be "LSCO".
<code>SessionId</code>		Required	The id from the Logon reply. Required on every message after logon.
<code>MsgType</code>	<code>OrderReplaceMultileg</code>	Required	
<code>ClientOrderID</code>		Required	A new id for the replacing order: 16 to 32 characters, unique within the trading day.
<code>OrderQty</code>		Required	The new quantity.
<code>Price</code>		Required	The new net price for the spread.
<code>OrigClientOrderID</code>		Required	The ClientOrderID on the multi-leg order you want to replace.
<code>Account</code>		Optional	

Responses

OrderReplaceMultilegUpdate

A successful replace is acknowledged by an **OrderReplaceMultilegUpdate** echoing the request with `ErrorCode: 0` and `ErrorText: "Validated OK"`. `OrderStatus` carries the order's state at the time of the acknowledgement, and the server-rebuilt legs ride on the echo. The original order then reports `PENDING_REPLACE` through **OrderMultilegUpdate** messages carrying per-leg statuses.

```

{
  "Account": "MyAccount",
  "ClientOrderID": "6f0b2d4a8c3e51f7b9d1e3a5c7f9b1d3",
  "OrderQty": "8",
  "OrderType": "LIMIT",
  "OrigClientOrderID": "b2d7e9f4a1c6483e8d0f2a6c4e8b1d75",
  "Price": "0.24",
  "Rule80A": "AGENCY_SINGLE_ORDER",
  "Side": "SELL",
  "Symbol": "BAC",
  "TimeInForce": "DAY",
  "ExchangeDestination": "SMART",
  "ClientID": "MyClientID",
  "SecurityType": "MULTI_LEG_INSTRUMENT",
  "NumLegs": [
    {
      "LegSymbol": "BAC",
      "LegPositionEffect": "CLOSE",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "510.00",
      "LegRatioQty": "1",
      "LegSide": "BUY",
      "LegRefID": "2-365297671"
    },
    {
      "LegSymbol": "BAC",
      "LegPositionEffect": "CLOSE",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "508.00",
      "LegRatioQty": "1",
      "LegSide": "SELL",
      "LegRefID": "1-365297670"
    }
  ],
  "OptOrLS": "LSConnect",
  "ErrorCode": 0,
  "ErrorText": "Validated OK",
  "SendingTime": "20260826-16:40:49.045",
  "OrderStatus": "NEW",
  "MsgType": "OrderReplaceMultilegUpdate"
}

```

OrderCancelReject

A failure returns an **OrderCancelReject** with `CancelRejectResponseTo` set to `ORDER_REPLACE`. This example shows a replace that arrived after the order had already completed:

```
{
  "MsgType": "OrderCancelReject",
  "ClientID": "MyClientID",
  "ClientOrderID": "6f0b2d4a8c3e51f7b9d1e3a5c7f9b1d3",
  "OrigClientOrderID": "b2d7e9f4a1c6483e8d0f2a6c4e8b1d75",
  "OrderID": "0-2163475118",
  "OrderStatus": "REJECTED",
  "CancelRejectResponseTo": "ORDER_REPLACE",
  "ErrorCode": 203,
  "ErrorText": "##:REJECT - Order not active ClientOrderID b2d7e9f4a1c6483e8d0f2a6c4e8b1d75",
  "SendingTime": "20260814-10:47:12.688"
}
```

API REFERENCE

Request account update

Request the account's current balances and positions by sending an AccountUpdate message.

Request

```
{
  "ClientID": "LSCO",
  "Account": "MyAccount",
  "MsgType": "AccountUpdate",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Request Properties

Property	Value	Required	Description
ClientID		Required	In production, the "ClientID" field will be "LSCO".
Account		Required	
SessionId		Required	The id from the Logon reply. Required on every message after logon.
MsgType	AccountUpdate	Required	

Responses

Returns an [AccountBalanceStatus](#) and a [PositionStatus](#) update message.

In paper trading, the request can succeed without returning a balance snapshot, because balance and risk data is not always available there.

API REFERENCE

Request a locate quote

See [Short a hard-to-borrow symbol](#) for the full flow.

Send a **LocateRequest** message to get a borrow rate quote for the desired symbol and quantity.

Request

```
{
  "ClientID": "LSC0",
  "Account": "MyAccount",
  "MsgType": "LocateRequest",
  "Symbol": "ZVZZT",
  "RequestQty": "100",
  "LocateType": "L",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Request Properties

Property	Value	Required	Description
MsgType	LocateRequest	Required	
ClientID		Required	Your client identifier
Account		Required	The account requesting the locate
SessionId		Required	The id from the Logon reply. Required on every message after logon.
Symbol		Required	Ticker symbol of the HTB security
RequestQty		Required	Number of shares to locate
LocateType	L	Required	A standard locate (L). The long form LOCATE is also accepted.
QuoteRequestID		Optional	Identifies the quote request. The shipped clients generate a 32-character id when not supplied.

Responses

LocateQuote

The quote arrives as a **LocateQuote** message carrying the borrow rate in `OfferPx` (price per share), the available quantity in `OfferSize`, and your `QuoteRequestID`. Review it before proceeding, as rates on HTB symbols can vary significantly based on availability.

```
{
  "QuoteRequestID": "a35b71c3b3d14303a735442ddf323dc",
  "QuoteID": "a35b71c3b3d14303a735442ddf323dc",
  "Symbol": "ZVZZT",
  "SymbolSfx": null,
  "OfferPx": "0",
  "OfferSize": "0",
  "Account": "MyAccount",
  "ClientID": "MyClientID",
  "MsgType": "LocateQuote",
  "LocateType": "LOCATE",
  "SendingTime": "20260826-16:44:43.940"
}
```

LocateRequestReject

A failed request returns a **LocateRequestReject** with your `QuoteRequestID`, an `ErrorCode` and `ErrorText`; see [Error Codes](#) for the locate code ranges.

API REFERENCE

Submit a locate order

See [Short a hard-to-borrow symbol](#) for the full flow.

If the rate is acceptable, submit a **LocateOrder** message using the quoted price to secure the borrow. The **Price** field must reflect the rate returned in the quote response.

Request

```
{
  "ClientID": "LSCO",
  "Account": "MyAccount",
  "MsgType": "LocateOrder",
  "Symbol": "ZVZZT",
  "RequestQty": "100",
  "LocateType": "L",
  "Price": "0.06613",
  "SessionId": "54c5dd04-af37-48d4-9098-e8a3d85ce260"
}
```

Request Properties

Property	Value	Required	Description
MsgType	LocateOrder	Required	
ClientID		Required	Your client identifier
Account		Required	The account requesting the locate
SessionId		Required	The id from the Logon reply. Required on every message after logon.
Symbol		Required	Ticker symbol of the HTB security
RequestQty		Required	Number of shares to locate
LocateType	L	Required	A standard locate (L). The long form LOCATE is also accepted.
Price		Required	Use the rate returned from the quote
ClientOrderID		Optional	Identifies the locate order. The shipped clients generate a 32-character id when not supplied.

Responses

The confirmation arrives as a **LocateExecution** message carrying your `ClientOrderID`. Once it confirms the borrow, you can submit your **SELL_SHORT** order for up to the located quantity. Locates are valid for the current trading day and expire at end of day.

A failed locate order returns a **LocateOrderReject** with your `ClientOrderID`, an `ErrorCode` (the broker's reject reason plus 1430) and `ErrorText`.

API REFERENCE

Execution reports

This message is sent by the server; there is no request to make.

Sent when

Execution reports are generated in response to changes in the state of the order.

Message

```
{
  "MsgType": "ExecutionReport",
  "Account": "MyAccount",
  "AvgPrice": "20.5",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "CumQty": "100",
  "ExecID": "E-2163476053",
  "ExecTransType": "NEW",
  "LastPx": "20.5",
  "LastShares": "100",
  "OrderID": "O-2163475242",
  "OrderQty": "100",
  "OrderStatus": "FILLED",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "GOOG",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "ExecType": "FILL",
  "LeavesQty": "0",
  "SecurityType": "COMMON_STOCK",
  "RefClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "ErrorCode": 0,
  "SendingTime": "20260826-16:15:34.546"
}
```

Properties

Property	Type	Description
<code>MsgType</code>	string	ExecutionReport
<code>Account</code>	string	Echoed from the order.
<code>AvgPrice</code>	string	Average fill price so far.
<code>ClientOrderID</code>	string	The id of the order this report belongs to.
<code>CumQty</code>	string	Total quantity filled so far.
<code>ExecID</code>	string	The broker's execution id. Values starting <code>##:</code> were generated by the API, not the broker.
<code>ExecTransType</code>	string	Observed value: <code>NEW</code> . See Enums .
<code>LastPx</code>	string	Price of this execution.
<code>LastShares</code>	string	Shares in this execution.
<code>OrderID</code>	string	The broker's order id. <code>NONE</code> when the broker has not assigned one.
<code>Price</code>	string	Echoed from the order, on limit orders.
<code>OrderQty</code>	string	Echoed from the order.
<code>OrderStatus</code>	string	The order's state after this execution. See Enums .
<code>OrderType</code>	string	Echoed from the order.
<code>Side</code>	string	Echoed from the order.
<code>Symbol</code>	string	Echoed from the order.
<code>TimeInForce</code>	string	Echoed from the order.
<code>ExchangeDestination</code>	string	Echoed from the order.
<code>ClientID</code>	string	Echoed from the order.
<code>ExecType</code>	string	Observed value: <code>FILL</code> . See Enums .
<code>LeavesQty</code>	string	Quantity remaining open.
<code>SecurityType</code>	string	Instrument type. Observed values are on Enums .
<code>OrigClientOrderID</code>	string	Present after a replace: the id of the order that was replaced.
<code>RefClientOrderID</code>	string	The original ClientOrderID of the order chain. It stays constant across replaces, while <code>ClientOrderID</code> changes with each one.

ErrorCode number **0** on a normal report.

SendingTime string Server timestamp in UTC, **YYYYMMDD-HH:MM:SS.sss**.

On option executions, the report also identifies the contract: **MaturityYearMonth**, **MaturityDay**, **PutOrCall** and **StrikePrice**. On the legs of a multi-leg order, each execution reports one leg, marked **MultiLegReportingType: "INDIVIDUAL_LEG_OF_A_MULTI_LEG_SECURITY"** with the leg's **LegRefID**. Executions can also carry routing details from the broker, such as **LastMkt**, **ExecBroker** and **ContraBroker**.

API REFERENCE

Order status

This message is sent by the server; there is no request to make.

Sent when

Sent when an order changes state, carrying the order's current status.

At connect the same picture arrives as **OrderSingleStatus** messages, one per open order, before the **Ligon** reply. An **OrderSingleStatus** carries the same fields as an **OrderSingleUpdate**; only the **MsgType** differs. Open multi-leg orders arrive as **OrderMultilegStatus**, carrying the fields of a [multi-leg order](#) instead. See [Connection lifecycle](#).

Message

```
{
  "Account": "MyAccount",
  "AvgPrice": "20.5",
  "ClientOrderID": "9d1c8e2ab6f2409caf39f4f2b1c8a7d3",
  "CumQty": "100",
  "LastPx": "20.5",
  "LastShares": "100",
  "OrderID": "0-2163475242",
  "OrderQty": "100",
  "OrderStatus": "FILLED",
  "OrderType": "LIMIT",
  "Price": "20.5",
  "Side": "BUY",
  "Symbol": "GOOG",
  "TimeInForce": "DAY",
  "ExchangeDestination": "NSDQ",
  "ClientID": "MyClientID",
  "LeavesQty": "0",
  "SecurityType": "COMMON_STOCK",
  "NotHeld": "TRUE",
  "ErrorCode": 0,
  "ErrorText": "",
  "SendingTime": "20260826-16:15:34.547",
  "MsgType": "OrderSingleUpdate"
}
```

Properties

Property	Type	Description
Account	string	Echoed from the order.
AvgPrice	string	Average fill price so far.
ClientOrderID	string	The id of the order this update belongs to.
CumQty	string	Total quantity filled so far.
LastPx	string	Price of the last execution.
LastShares	string	Shares in the last execution.
OrderID	string	The broker's order id. NONE when the broker has not assigned one.
OrderQty	string	Echoed from the order.
Price	string	Echoed from the order, on limit orders.
OrderStatus	string	The order's current state. See Enums .
OrderType	string	Echoed from the order.
Side	string	Echoed from the order.
Symbol	string	Echoed from the order.
TimeInForce	string	Echoed from the order.
ExchangeDestination	string	Echoed from the order.
ClientID	string	Echoed from the order.
LeavesQty	string	Quantity remaining open.
SecurityType	string	Instrument type. Observed values are on Enums .
NotHeld	string	Marks the order not held. Applied by the server as a default of TRUE when the request does not set it.
ErrorCode	number	0 on a normal update.
ErrorText	string	Empty on a normal update.
SendingTime	string	Server timestamp in UTC, YYYYMMDD-HH:MM:SS.sss .
MsgType	string	OrderSingleUpdate

API REFERENCE

Cancel rejections

This message is sent by the server; there is no request to make.

Sent when

An unsuccessful **OrderCancel** returns an **OrderCancelReject** with **CancelRejectResponseTo** set to **ORDER_CANCEL**. Failure to update an order with **OrderReplace** returns one with **CancelRejectResponseTo** set to **ORDER_REPLACE**.

Message

```
{
  "MsgType": "OrderCancelReject",
  "ClientOrderID": "5c9d1e7f3a2b46c8a0d4f6e8b2c4a6d0",
  "OrderID": "0-2163475290",
  "OrderStatus": "NEW",
  "OrigClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41",
  "ErrorText": "Cancel request rejected during T5",
  "CancelRejectReason": "BROKER_OPTION",
  "ClientID": "MyClientID",
  "CancelRejectResponseTo": "ORDER_CANCEL",
  "ErrorCode": 1300,
  "SendingTime": "20260826-16:30:37.933",
  "RefClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41"
}
```

Properties

Property	Type	Description
<code>MsgType</code>	string	OrderCancelReject
<code>ClientID</code>	string	Echoed from the request.
<code>ClientOrderID</code>	string	The id of the cancel or replace request that failed.
<code>OrigClientOrderID</code>	string	The id of the order the request targeted.
<code>RefClientOrderID</code>	string	The original id of the order chain.
<code>OrderID</code>	string	The broker's order id.
<code>OrderStatus</code>	string	The order's current state. The order lives on with this status after the rejection.
<code>CancelRejectResponseTo</code>	string	<code>ORDER_CANCEL</code> when rejecting a cancel, <code>ORDER_REPLACE</code> when rejecting a replace.
<code>CancelRejectReason</code>	string	The broker's rejection reason category.
<code>ErrorCode</code>	number	The reason for the rejection. See Error Codes .
<code>ErrorText</code>	string	Human-readable reason.
<code>SendingTime</code>	string	Server timestamp in UTC, <code>YYYYMMDD-HH:MM:SS.sss</code> .

After the rejection, an **OrderSingleUpdate** follows, carrying the order's unchanged state and the rejection reason in its `ErrorText`.

API REFERENCE

Positions

These messages are sent by the server; there is no request to make.

Positions arrive as two messages. **PositionStatus** is the snapshot sent at connect, carrying a `positions` array of every open position. **PositionUpdate** carries a single `position` object each time one position changes.

PositionStatus

Sent at connect, before the application starts trading.

```
{
  "positions": [
    {
      "acct": "MyAccount",
      "creditUsed": 0,
      "dailyPL": -0.3,
      "flags": 0,
      "floatingBpUsed": 0,
      "groupId": "LSC0",
      "instrument": "E",
      "openOrderCreditUsed": 0,
      "pos": 5,
      "posAvgPrice": 1.85,
      "realPL": 0,
      "sym": "PRQR",
      "unrealPL": -0.3,
      "valueBought": 0,
      "valueSold": 0
    }
  ],
  "MsgType": "PositionStatus",
  "ErrorCode": 0,
  "ErrorText": "",
  "SendingTime": "20260814-10:47:22.842"
}
```

PositionUpdate

Sent when a position changes. Contains the current position quantity, position average price, and notional value per symbol.

```

{
  "MsgType": "PositionUpdate",
  "ErrorCode": 0,
  "position": {
    "acct": "MyAccount",
    "groupId": "MyClientID",
    "sym": "IBM",
    "instrument": "E",
    "pos": 0,
    "posAvgPrice": 0,
    "creditUsed": 0,
    "openOrderCreditUsed": 0,
    "floatingBpUsed": 0,
    "realPL": 0,
    "unrealPL": 0,
    "dailyPL": 0,
    "flags": 0
  }
}

```

Position fields

Both messages carry the position object: `PositionStatus` as a `positions` array, `PositionUpdate` as a single `position` object. Updates sent immediately after an execution carry the position quantity with the analytic fields (`posAvgPrice` , the profit-and-loss and credit fields) as `0` , omit `valueBought` and `valueSold` , and have no top-level `ErrorText` or `SendingTime` ; updates published by the account system carry the full values and the complete top level.

Property	Type	Description
<code>acct</code>	string	The account.
<code>creditUsed</code>	number	
<code>dailyPL</code>	number	Profit and loss for the day.
<code>flags</code>	number	
<code>floatingBpUsed</code>	number	
<code>groupId</code>	string	Carries the ClientID in observed payloads.
<code>instrument</code>	string	Observed value: <code>E</code> . See Enums .
<code>openOrderCreditUsed</code>	number	
<code>pos</code>	number	Current position quantity.
<code>posAvgPrice</code>	number	Position average price.
<code>realPL</code>	number	Realized profit and loss.
<code>sym</code>	string	Ticker symbol.
<code>unrealPL</code>	number	Unrealized profit and loss.
<code>valueBought</code>	number	
<code>valueSold</code>	number	

API REFERENCE

Account Balance

These messages are sent by the server; there is no request to make.

Balances arrive as two messages, the same way positions do. **AccountBalanceStatus** is the snapshot. **AccountBalanceUpdate** carries a change.

AccountBalanceStatus

Sent at connect, and returned in response to an [AccountUpdate](#) request. Provides Buying Power Updates and account balance information.

```

{
  "account_balances": {
    "PDT": 0,
    "SMA": 0,
    "acct": "MyAccount",
    "acctClass": "\u0000",
    "credit": 1540.12,
    "creditMultiplier": 1,
    "creditRemaining": 1540.12,
    "dayTradeBuyingPower": 6160.48,
    "dayTradeOvernightRegTBuyingPower": 1539.82,
    "excess": 1549.07,
    "grossMargin": 0,
    "groupId": "LSC0",
    "heldBackFunds": 0,
    "maintReq": 0,
    "mktValue": 13.75,
    "netBuyingPower": 1540.12,
    "optionBuyingPower": 1540.12,
    "pdtCreditRemaining": 0,
    "pendingOrdersCount": 0,
    "pendingOrdersMarginRequirements": 0,
    "pendingPDT": 0,
    "pendingSMA": 0,
    "realizedPL": 0,
    "settledFunds": 1540,
    "smaCreditRemaining": 0,
    "startOfDayCash": 1540,
    "stockBuyingPower": 1540.12,
    "ttlEquity": 1549.07,
    "unRealizedPL": -0.3,
    "unsettledFunds": 0,
    "valueBought": 0,
    "valueSold": 0
  },
  "MsgType": "AccountBalanceStatus",
  "ErrorCode": 0,
  "ErrorText": "",
  "SendingTime": "20260814-10:47:22.842"
}

```

AccountBalanceUpdate

Sent when balances or risk figures change. Updates can also arrive between changes. The values are native JSON numbers. The update carries a single `account_balances` object, wrapped the same way as the snapshot.

Balance fields

Property	Type	Description
<code>account_balances.PDT</code>	number	
<code>account_balances.SMA</code>	number	
<code>account_balances.acct</code>	string	The account.
<code>account_balances.acctClass</code>	string	Observed value: a null character. See Enums .
<code>account_balances.credit</code>	number	
<code>account_balances.creditMultiplier</code>	number	
<code>account_balances.creditRemaining</code>	number	
<code>account_balances.dayTradeBuyingPower</code>	number	Day trade buying power.
<code>account_balances.dayTradeOvernightRegTBuyingPower</code>	number	
<code>account_balances.excess</code>	number	
<code>account_balances.grossMargin</code>	number	
<code>account_balances.groupId</code>	string	Carries the ClientID in observed payloads.
<code>account_balances.heldBackFunds</code>	number	
<code>account_balances.maintReq</code>	number	Maintenance requirement.
<code>account_balances.mktValue</code>	number	
<code>account_balances.netBuyingPower</code>	number	Net buying power.
<code>account_balances.optionBuyingPower</code>	number	Option buying power.
<code>account_balances.pdtCreditRemaining</code>	number	
<code>account_balances.pendingOrdersCount</code>	number	Number of pending orders.
<code>account_balances.pendingOrdersMarginRequirements</code>	number	Margin required by pending orders.
<code>account_balances.pendingPDT</code>	number	
<code>account_balances.pendingSMA</code>	number	
<code>account_balances.realizedPL</code>	number	Realized profit and loss.

<code>account_balances.settledFunds</code>	number	Settled funds.
<code>account_balances.smaCreditRemaining</code>	number	
<code>account_balances.startOfDayCash</code>	number	Cash at the start of the day.
<code>account_balances.stockBuyingPower</code>	number	Stock buying power.
<code>account_balances.ttlEquity</code>	number	Total equity.
<code>account_balances.unRealizedPL</code>	number	Unrealized profit and loss.
<code>account_balances.unsettledFunds</code>	number	Unsettled funds.
<code>account_balances.valueBought</code>	number	
<code>account_balances.valueSold</code>	number	
<code>MsgType</code>	string	AccountBalanceStatus
<code>ErrorCode</code>	number	0 on a normal message.
<code>ErrorText</code>	string	Empty on a normal message.
<code>SendingTime</code>	string	Server timestamp in UTC, <code>YYYYMMDD-HH:MM:SS.sss</code> .

API REFERENCE

Enums

Every enumerated value the documentation currently defines, in one place, verified against the backend message specification.

Values you send

Field	Values	Notes
<code>Side</code>	<code>BUY</code> , <code>SELL</code> , <code>SELL_SHORT</code>	
<code>TimeInForce</code>	<code>DAY</code> , <code>GOOD_TILL_CANCEL</code> , <code>AT_THE_OPENING</code> , <code>IMMEDIATE_OR_CANCEL</code>	Defaults to <code>DAY</code> . Accounts may be restricted to a subset.
<code>OrderType</code>	<code>MARKET</code> , <code>LIMIT</code> , <code>STOP</code> , <code>STOP_LIMIT</code> , <code>MARKET_ON_CLOSE</code> , <code>LIMIT_ON_CLOSE</code>	Market/limit-on-open orders use <code>TimeInForce</code> <code>AT_THE_OPENING</code> .
<code>ExecInstruction</code>	<code>TRAILING_STOP_PEG</code>	With <code>PegDifference</code> , for trailing stops.
<code>OpenClose</code>	<code>O</code> , <code>C</code>	Opening or closing a position.
<code>PutOrCall</code>	<code>PUT</code> , <code>CALL</code>	
<code>LocateType</code>	<code>LOCATE</code>	Locate messages.
<code>LegCFIcode</code>	<code>CALL_OPTION</code> , <code>PUT_OPTION</code> , <code>COMMON_STOCK</code>	Multi-leg legs.
<code>LegSide</code>	<code>BUY</code> , <code>SELL</code>	Multi-leg legs.
<code>LegPositionEffect</code>	<code>OPEN</code> , <code>CLOSE</code>	Multi-leg legs.
<code>ExchangeDestination</code>	<code>SMART</code> plus one value per venue	Available Routes.

Values you receive

Field	Values	Notes
OrderStatus	PENDING_NEW, NEW, PARTIALLY_FILLED, FILLED, DONE_FOR_DAY, CANCELED, REPLACED, PENDING_CANCEL, PENDING_REPLACE, STOPPED, REJECTED, SUSPENDED, EXPIRED	
ExecType	NEW, PARTIAL_FILL, FILL, DONE_FOR_DAY, CANCELED, REPLACE, PENDING_CANCEL, PENDING_REPLACE, STOPPED, REJECTED, SUSPENDED, PENDING_NEW, EXPIRED	
ExecTransType	NEW, CANCEL, CORRECT, STATUS	
CancelRejectResponseTo	ORDER_CANCEL, ORDER_REPLACE	OrderCancelReject .
SecurityType	COMMON_STOCK, OPTION, MLEG	As seen on order messages and reports.
NotHeld	TRUE	Server-applied default on orders.
instrument	E, 0	Positions: equity or option. Other asset types arrive as their raw type code.
acctClass	a null character ()	Account balances; originates upstream.

API REFERENCE

Error Codes

Every message the server sends carries an `ErrorCode`, and most also carry an `ErrorText`. `0` means success.

```
{
  "MsgType": "OrderSingleUpdate",
  "ErrorCode": 0,
  "ErrorText": "",
  "SendingTime": "20260814-10:38:24.722"
}
```

Note that `ErrorCode` is a JSON number, not a string, unlike prices and quantities, which are strings throughout. See [JSON Data Model](#).

One error at a time

NOTE

If a request contains several errors, the API reports **one error at a time**. For example, if an order has both an invalid Symbol and an incorrect Price format, the response might only highlight the Symbol issue; after correcting that, a subsequent submission could then fail due to the Price error.

We recommend thoroughly validating your JSON messages against the API specifications (field formats, required tags, and valid values) before submission, to minimise failed attempts. Future API updates may enhance this to return all validation errors in a single response. In the meantime, use the [API Console](#) to test orders and review error messages, as it provides real-time feedback to help identify issues iteratively.

Rejections

A failed cancel or replace comes back as a distinct message rather than an error code. See [OrderCancelReject](#).

Codes

`ErrorText` carries the specific reason; the code gives the category.

Code	Meaning
0	Success.
1	Message could not be parsed: bad JSON, an empty object, or a frame over 9999 bytes.
2	<code>MsgType</code> missing.
3	<code>Symbol</code> missing.
6	<code>Side</code> missing.
7	<code>ClientOrderID</code> missing.
9	<code>OrigClientOrderID</code> missing.
18	<code>Price</code> missing or invalid.
19	<code>StopPrice</code> missing or invalid.
20	<code>OrderQty</code> missing or invalid.
23	<code>PegDifference</code> missing on a trailing stop.
25	<code>ClientOrderID</code> over 32 characters.
32	<code>ClientID</code> missing.
33	<code>ClientID</code> invalid.
35	Not allowed for the account, for example a restricted <code>TimeInForce</code> .
37	<code>ClientOrderID</code> under 16 characters.
60	<code>Account</code> missing.
100	<code>MsgType</code> not recognised.
118	<code>ApiKey</code> missing on logon.
120	Invalid <code>ApiKey</code> , or the key already has a live connection. <code>ErrorText</code> says which.
160	A replace may change <code>OrderType</code> only from LIMIT to MARKET.
161	<code>SecurityType</code> not supported for the account.
162	<code>Symbol</code> invalid.
200	Duplicate <code>ClientOrderID</code> .
201	Maximum <code>OrderQty</code> exceeded.

202	Position limit exceeded.
203	The order's state does not allow the request.
204	Sell order mismarked.
205	<code>OrderQty</code> cannot be increased on a replace.
206	<code>SecurityExchange</code> invalid.
207	Option definition incomplete: options need <code>Symbol</code> , <code>OpenClose</code> , <code>MaturityYearMonth</code> , <code>PutOrCall</code> , <code>StrikePrice</code> and <code>MaturityDay</code> .
299	Session disconnected; log on again. Broadcast when the engine restarts.
300	<code>SessionId</code> validation failed. The server closes the socket.
301	<code>SessionId</code> missing.
351	Trading is disconnected upstream. On a logon reply this is still a successful logon; orders reject with the same code until trading recovers.
400	Internal error.
404	Order not found.
500	Multi-leg side restriction (broker-dependent).
501	Multi-leg <code>SecurityType</code> must be <code>MLEG</code> .
502	Multi-leg legs missing.
503	Leg <code>LegCFIcode</code> missing.
504	Leg contract multiplier error.
505	Leg security exchange missing.
506	Leg maturity date missing on an option leg.
1101 - 1135	Message failed schema validation: unknown field, unknown value, or bad format. <code>ErrorText</code> names the field.
1401 - 1404	Locate service unavailable or errored.
1411	Locate quote request invalid.
1412	<code>QuoteRequestID</code> missing.
1421	Locate order request invalid.

1431 and up Locate order rejected; the code is the broker's reject reason plus 1430.

The server closes the socket after codes 118, 120 and 300; every other error leaves the connection open.

API REFERENCE

Symbology

Equities

Equity orders are routed using the ticker symbol (`Symbol`). Lightspeed Connect uses the NYSE Consolidated Quote System (CQS) symbology for US equities. Common stocks use standard ticker symbols. Other equity classes use suffixes to indicate the security type.

Type	Format	Example
Common Stock	<code>Symbol</code>	<code>IBM</code>
Class Stock	<code>Symbol/Class</code>	<code>IBM/B</code>
Preferred Stock	<code>Symbolp</code>	<code>IBMp</code>
Preferred Class	<code>SymbolpClass</code>	<code>IBMpA</code>
Preferred + Called/Redeemed	<code>SymbolpClass/CL</code>	<code>IBMpA/CL</code>
Preferred + Convertible	<code>SymbolpClass/CV</code>	<code>IBMpA/CV</code>
Rights	<code>Symbolr</code>	<code>IBMr</code>
Rights When Issued	<code>Symbolrw</code>	<code>IBMrw</code>
Warrants	<code>Symbol/WS</code>	<code>IBM/WS</code>
Warrants When Issued	<code>Symbol/WSw</code>	<code>IBM/WSw</code>
Warrants When Distributed	<code>Symbol/WD</code>	<code>IBM/WD</code>
When Issued	<code>Symbolw</code>	<code>IBMw</code>
Unit	<code>Symbol/u</code>	<code>IBM/u</code>
Class Warrants	<code>Symbol/WS/Class</code>	<code>IBM/WS/A</code>

For most use cases, you will be working with common stock symbols (for example `AAPL` , `MSFT` , `TSLA`). The suffixed formats apply when trading or viewing positions in preferred shares, warrants, rights, or other specialized equity classes.

Options

- Single-leg options

- Multi-leg options

Options orders are routed by specifying the root symbol (`Symbol`), followed by the standard fields `MaturityYearMonth`, `PutOrCall`, `StrikePrice` and `MaturityDay`. Where similar contracts exist on the same underlying or trading class with the same strike, type and expiration but different multipliers, `ContractMultiplier` identifies the contract.

OSI Symbol Mapping

The OCC Options Symbology Initiative (OSI) encodes a contract as the root symbol, a `YYMMDD` expiration date, `C` or `P`, and the strike price multiplied by 1,000, padded to eight digits. The table below maps OSI symbols to the equivalent order fields for the Lightspeed Connect API:

OSI Symbol	Symbol	MaturityYearMonth	MaturityDay	PutOrCall	StrikePrice
NKE220924C00099000	NKE	202209	24	CALL	99.00
NKE220924P00099000	NKE	202209	24	PUT	99.00
TSLA251031C00437500	TSLA	202510	31	CALL	437.50
SPY241018C00430000	SPY	202410	18	CALL	430.00

For example, the contract `TSLA251031C00437500` is specified on an order as:

```
{
  "Symbol": "TSLA",
  "MaturityYearMonth": "202510",
  "MaturityDay": "31",
  "PutOrCall": "CALL",
  "StrikePrice": "437.50"
}
```

NOTE

`Symbol` takes the root ticker only, for example `TSLA`, not the full OSI string.

For a payload per order type, see [Code Examples](#). Additional order types are available on request.

API REFERENCE

Available Routes

This section lists the available routes for directing orders in LS Connect.

To use a route, send the table's **ExchangeDestination Value** in the `ExchangeDestination` field of your order. For example, to route an order to the NASDAQ exchange, set `"ExchangeDestination": "NSDQ"`.

The value is an internal routing identifier. Equity orders default to `SMART` when no destination is entered.

LS Connect Pro Equity Routes

Route	ExchangeDestination Value	Description
AMEX	AMEX	American Stock Exchange
ARCA	NYSEARCA	NYSE Arca (electronic exchange)
BATS	BATS	BATS Global Markets
BATY	BATYSLIM	BATS Y-Exchange (slim routing)
EDGA	EDGAROUT	EDGA Exchange (Direct Edge)
EDGX	EDGXROUT	EDGA Exchange (Direct Edge)
NSDQ	NSDQ	NASDAQ Stock Market
NYSE	NYSE	New York Stock Exchange

LS Connect Pro Options Routes

Route	ExchangeDestination Value	Description
SMART	SMART	Smart routing (multi-venue)

LS Connect Lite Equity Routes

Route	ExchangeDestination Value	Description
LSPT	PDQSMART	PDQ Smart Routing (pooled)
MNGD (APEX)	MNGD	Managed routing via Apex
ARCA	ARCA	NYSE Arca (direct)

LS Connect Lite Options Routes

Route	ExchangeDestination Value	Description
SMART	SMART	Smart routing (multi-venue)

Note: SMART routing automatically selects the best available venue for options execution.

PART V

Code Examples

Limit

Limit On Open

Limit On Close

Market

Market On Open

Market On Close

Stop

Stop-Limit

Trailing-Stop

Single Leg Call

Single Leg Put

Credit Spread

Debit Spread

Cancel Order

Cancel Replace Order

Account Update

Locate Quote Request

Locate Order

CODE EXAMPLES

Limit

A Limit order is an order to buy or sell at a specified price or better. The Limit order ensures that if the order fills, it will not fill at a price less favorable than your limit price, but it does not guarantee a fill. Sent as an [OrderSingle](#).

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "AMC",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "LIMIT",
  "OrderQty": "100",
  "TimeInForce": "DAY",
  "Price": "2.60"
}
```

CODE EXAMPLES

Limit On Open

A Limit on Open (LOO) order is typically used by investors who try to execute a trade at the start of the day, but want to limit the price they pay. Sent as an [OrderSingle](#).

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "AMC",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "LIMIT",
  "OrderQty": "100",
  "TimeInForce": "AT_THE_OPENING",
  "Price": "2.60"
}
```

The combination is `OrderType: "LIMIT"` with `TimeInForce: "AT_THE_OPENING"`.

CODE EXAMPLES

Limit On Close

A Limit on Close (LOC) order will be submitted at the close and will execute if the closing price is at or better than the submitted limit price. Sent as an [OrderSingle](#).

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "AMC",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "LIMIT_ON_CLOSE",
  "OrderQty": "100",
  "TimeInForce": "DAY",
  "Price": "2.60"
}
```

CODE EXAMPLES

Market

Market order is an order to buy or sell at the market bid or offer price. Sent as an [OrderSingle](#).

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "AMC",
  "ExchangeDestination": "NSDQ",
  "Side": "BUY",
  "OrderType": "MARKET",
  "OrderQty": "100",
  "TimeInForce": "DAY"
}
```

CODE EXAMPLES

Market On Open

A Market on Open (MOO) combines a market order with a TimeInForce to create an order that is automatically submitted at the market's open and fills at the market price. Sent as an OrderSingle.

```
{  
  "ClientID": "MyClientID",  
  "Account": "MyAccount",  
  "MsgType": "OrderSingle",  
  "Symbol": "AMC",  
  "ExchangeDestination": "NSDQ",  
  "Side": "BUY",  
  "OrderType": "MARKET",  
  "TimeInForce": "AT_THE_OPENING",  
  "OrderQty": "100"  
}
```

The combination is `OrderType: "MARKET"` with `TimeInForce: "AT_THE_OPENING"`.

CODE EXAMPLES

Market On Close

A Market on Close (MOC) order is a market order that is submitted to execute as close to the closing price as possible. Sent as an [OrderSingle](#).

```
{  
  "ClientID": "MyClientID",  
  "Account": "MyAccount",  
  "MsgType": "OrderSingle",  
  "Symbol": "AMC",  
  "ExchangeDestination": "NSDQ",  
  "Side": "BUY",  
  "OrderType": "MARKET_ON_CLOSE",  
  "OrderQty": "100",  
  "TimeInForce": "DAY",  
  "Price": "2.60"  
}
```

CODE EXAMPLES

Stop

A Stop order is an instruction to submit a buy or sell market order if and when the user-specified stop trigger price is attained or penetrated. Sent as an [OrderSingle](#).

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "AMC",
  "ExchangeDestination": "EDGAROUT",
  "Side": "BUY",
  "OrderType": "STOP",
  "StopPrice": "2.60",
  "OrderQty": "100",
  "TimeInForce": "DAY"
}
```

StopPrice is required; it sets the trigger price.

CODE EXAMPLES

Stop-Limit

A StopLimit (SL) order is an instruction to submit a buy or sell limit order when the user-specified stop trigger price is attained or penetrated. Sent as an [OrderSingle](#).

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "AMC",
  "ExchangeDestination": "EDGAROUT",
  "Side": "SELL",
  "OrderType": "STOP_LIMIT",
  "OrderQty": "100",
  "TimeInForce": "DAY",
  "StopPrice": "2.70",
  "Price": "2.60"
}
```

`StopPrice` sets the trigger and `Price` the limit; both are required.

CODE EXAMPLES

Trailing-Stop

A trailing stop is a stop order that moves with the market price by a fixed amount. Sent as an OrderSingle.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "AMC",
  "ExchangeDestination": "EDGAROUT",
  "Side": "SELL",
  "OrderType": "MARKET",
  "ExecInstruction": "TRAILING_STOP_PEG",
  "OrderQty": "100",
  "TimeInForce": "DAY",
  "PegDifference": "0.05"
}
```

The form is `OrderType: "MARKET"` with `ExecInstruction: "TRAILING_STOP_PEG"` and `PegDifference` set to the trailing amount.

CODE EXAMPLES

Single Leg Call

A call option is a derivatives contract giving the owner the right, but not the obligation, to buy a specified amount of an underlying security at a specified price within a specified time. Sent as an **OrderSingle**.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "GE",
  "Side": "BUY",
  "Currency": "USD",
  "OpenClose": "O",
  "OrderQty": "3",
  "TimeInForce": "DAY",
  "OrderType": "LIMIT",
  "Price": "0.45",
  "PutOrCall": "CALL",
  "StrikePrice": "510",
  "MaturityYearMonth": "202402",
  "MaturityDay": "16",
  "SecurityType": "OPTION",
  "ExchangeDestination": "SMART"
}
```

CODE EXAMPLES

Single Leg Put

A put option is a contract giving the option buyer the right, but not the obligation, to sell-or sell short-a specified amount of an underlying security at a predetermined price within a specified time frame. Sent as an [OrderSingle](#).

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderSingle",
  "Symbol": "DIS",
  "Side": "BUY",
  "Currency": "USD",
  "OpenClose": "O",
  "OrderQty": "4",
  "TimeInForce": "DAY",
  "OrderType": "LIMIT",
  "Price": "0.25",
  "PutOrCall": "PUT",
  "StrikePrice": "480",
  "MaturityYearMonth": "202402",
  "MaturityDay": "16",
  "SecurityType": "OPTION",
  "ExchangeDestination": "NSDQ"
}
```

CODE EXAMPLES

Credit Spread

A credit spread involves selling a high-premium option while purchasing a low-premium option in the same class or of the same security, resulting in a credit to the trader's account. Sent as an OrderMultileg.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderMultileg",
  "Symbol": "WORK",
  "Side": "SELL",
  "Currency": "USD",
  "OpenClose": "C",
  "OrderQty": "4",
  "TimeInForce": "DAY",
  "OrderType": "LIMIT",
  "Price": "-0.24",
  "SecurityType": "MLEG",
  "ExchangeDestination": "SMART",
  "NumLegs": [
    {
      "LegSymbol": "WORK",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "508.00",
      "LegRatioQty": "1",
      "LegSide": "SELL",
      "LegPositionEffect": "CLOSE"
    },
    {
      "LegSymbol": "WORK",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "510.00",
      "LegRatioQty": "1",
      "LegSide": "BUY",
      "LegPositionEffect": "CLOSE"
    }
  ]
}
```

CODE EXAMPLES

Debit Spread

A debit spread involves purchasing a high-premium option while selling a low-premium option in the same class or of the same security, resulting in a debit from the trader's account. Sent as an **OrderMultileg**.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderMultileg",
  "Symbol": "DIS",
  "Side": "BUY",
  "Currency": "USD",
  "OrderQty": "4",
  "TimeInForce": "DAY",
  "OrderType": "LIMIT",
  "Price": "0.30",
  "SecurityType": "MLEG",
  "ExchangeDestination": "SMART",
  "NumLegs": [
    {
      "LegSymbol": "DIS",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "508.00",
      "LegRatioQty": "1",
      "LegSide": "BUY",
      "LegPositionEffect": "OPEN"
    },
    {
      "LegSymbol": "DIS",
      "LegCFIcode": "CALL_OPTION",
      "LegSecurityType": "OPTION",
      "LegMaturityDate": "20240216",
      "LegStrikePrice": "510.00",
      "LegRatioQty": "1",
      "LegSide": "SELL",
      "LegPositionEffect": "OPEN"
    }
  ]
}
```

CODE EXAMPLES

Cancel Order

Cancels an open order. `OrigClientOrderID` names the order to cancel, and `ClientOrderID` is a new id for the cancel request itself. Both OrderSingle and OrderMultileg orders are canceled with this message. Sent as an **OrderCancel**.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderCancel",
  "ClientOrderID": "5c9d1e7f3a2b46c8a0d4f6e8b2c4a6d0",
  "OrigClientOrderID": "4b8e2f6d0a734c1e9b5d3a7c8f2e6b41"
}
```

CODE EXAMPLES

Cancel Replace Order

Replaces a resting order. `OrigClientOrderID` names the order to replace, and `ClientOrderID` is a new id for the replacing order. The alterable attributes are `OrderQty`, `Price`, `StopPrice` and `TimeInForce`. Sent as an OrderReplace.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "OrderReplace",
  "ClientOrderID": "8a4c6e2d0f9b41d7a3e5c7f9d1b3e5a7",
  "Symbol": "AMC",
  "OrderQty": "200",
  "Price": "2.60",
  "OrigClientOrderID": "7e3a9c5f1b8d4e6a2c4f8b0d6e2a9c53"
}
```

CODE EXAMPLES

Account Update

Requests the account's current balances and positions. The reply arrives as an AccountBalanceStatus and a PositionStatus message. Sent as an [AccountUpdate](#).

```
{  
  "ClientID": "MyClientID",  
  "Account": "MyAccount",  
  "MsgType": "AccountUpdate"  
}
```

CODE EXAMPLES

Locate Quote Request

Requests a borrow rate quote for the desired symbol and quantity. The quote arrives as a LocateQuote message. Sent as a [LocateRequest](#). See [Locates](#) for the full flow.

```
{  
  "ClientID": "MyClientID",  
  "Account": "MyAccount",  
  "MsgType": "LocateRequest",  
  "Symbol": "AAPL",  
  "RequestQty": "100",  
  "LocateType": "L"  
}
```

CODE EXAMPLES

Locate Order

Secures the borrow at the quoted rate. **Price** must carry the rate returned in the LocateQuote reply. Sent as a **LocateOrder**. See [Locates](#) for the full flow.

```
{
  "ClientID": "MyClientID",
  "Account": "MyAccount",
  "MsgType": "LocateOrder",
  "Symbol": "AAPL",
  "RequestQty": "100",
  "LocateType": "L",
  "Price": "0.06613"
}
```

PART VI

Tools

[Download the adapter](#)

[Using these docs with AI](#)

TOOLS

Download the adapter

The LS Connect Adapter is a Python client for the Lightspeed Connect WebSocket API. It logs on, answers heartbeats, stamps outgoing messages with the required session fields, and reconnects with exponential backoff.

The adapter is optional. The API is a plain WebSocket; see [Raw WebSocket](#) to implement a client directly.

Python

Requires Python 3.8 or later. Both archives contain identical files.

Format	Download
Zip	lightspeed-connect-python-client.zip
Gzipped tar	lightspeed-connect-python-client.tar.gz

[Setup and usage](#) covers dependency installation, run script configuration, and submitting a first order.

Credentials are read from the run configuration rather than from source. Moving between the [paper trading and production environments](#) requires no code change.

TOOLS

Using these docs with AI

These docs are written to be read by assistants as well as people. Every page is plain markdown at a stable URL, so you can hand any of it to ChatGPT, Claude, Cursor or your own agent without scraping HTML.

TIP: START HERE: LLMS.TXT AND LLMS-FULL.TXT

Two machine-readable files cover the entire documentation. [llms.txt](#) is an index with markdown links to every page, for assistants that discover and retrieve only the pages relevant to a request. [llms-full.txt](#) is the complete documentation in a single file, for tools that load the full context up front.

Every page is markdown

Take any documentation URL and read the `.md` file behind it. The path is the same:

```
# what you read in the browser
https://lightspeed-connect.connecttrade.com/developers/documentation/guides/adapter

# the same page as markdown
https://lightspeed-connect.connecttrade.com/docs/guides/adapter.md
```

These files need no authentication and are the same bytes the site renders. There is no second copy to drift out of date.

From any page

The rail on the right of every documentation page carries four actions:

Action	What it does
Copy page	Copies that page's raw markdown to your clipboard
View as Markdown	Opens the <code>.md</code> file directly
Open in ChatGPT	Opens ChatGPT with a prompt pointing at that page
Open in Claude	Opens Claude with a prompt pointing at that page

The two assistant links pass the markdown URL, not the page contents, so the assistant fetches the current version rather than a snapshot.

Writing code against the API

Give the assistant three things and it has everything it needs:

1. `llms.txt` for selective retrieval, or `llms-full.txt` for the complete documentation in one file. Both follow the [llms.txt convention](#). The [API Reference](#) has one page per message with request and response payloads.
2. **The adapter**, from [Download the adapter](#). It handles reconnection, heartbeats and `ClientOrderID` formatting, so generated code doesn't have to.
3. **Your environment**, from [Environments](#). Always start against paper trading, where orders are simulated.