Quickstart guide

Introduction

This guide demonstrates how to create and submit a Limit Order v4 using the @1inch/limit-order-sdk in a JavaScript environment. You'll learn how to:

- Configure your environment
- Approve token transfers
- Construct and sign a limit order
- Submit the signed order to the 1inch Orderbook API

Step 1: Install dependencies

Ensure that you have Node.js installed (v18 or higher is recommended).

In your project directory, initialize a new Node.js project and install the required libraries:

```
npm init -y
npm install @1inch/limit-order-sdk ethers
```

This will install:

- `@1inch/limit-order-sdk`: the core SDK for creating, signing, and submitting limit orders.
- `ethers`: a lightweight library for interacting with Ethereum.

Step 2: Set up wallet and network configuration

Create a new file, e.g., `limitOrder.js`, and begin by importing required libraries and initializing the wallet and provider.

```
import { Wallet, JsonRpcProvider, Contract } from "ethers";
import {
  LimitOrder,
  MakerTraits,
  Address,
  Api,
```

```
  getLimitOrderV4Domain,
} from "@1inch/limit-order-sdk";

// Standard ERC-20 ABI fragment (used for token approval)
const erc20AbiFragment = [
  "function approve(address spender, uint256 amount) external returns (bool)",
  "function allowance(address owner, address spender) external view returns (uint256)",
];

// Use environment variables to manage private keys securely
const privKey = process.env.PRIVATE_KEY;
const chainId = 1; // Ethereum mainnet

const provider = new JsonRpcProvider("https://cloudflare-eth.com/");
const wallet = new Wallet(privKey, provider);
```

Further, specify the assets, amounts, and expiration for your limit order.

```
const makerAsset = "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48"; // USDC
const takerAsset = "0x111111111117dc0aa78b770fa6a738034120c302"; // 1INCH

const makingAmount = 1_000_000n; // 1 USDC (in 6 decimals)
const takingAmount = 1_000_000_000_000_000_00n; // 1 1INCH (18 decimals)

const expiresIn = 120n; // seconds
const expiration = BigInt(Math.floor(Date.now() / 1000)) + expiresIn;
```

Step 3: Check allowance and approve the token

The 1inch Limit Order smart contract must be authorized to transfer the maker's tokens. This requires an ERC-20 `approve` transaction that grants allowance *per token per spender*—meaning a unique approval is needed for each token and for each contract address that will spend it.

Check the current allowance, and if it's insufficient, approve the required amount:

```
import { MaxUint256 } from "ethers";

const domain = getLimitOrderV4Domain(chainId);
const limitOrderContractAddress = domain.verifyingContract;

const makerAssetContract = new Contract(makerAsset, erc20AbiFragment, wallet);

const currentAllowance = await makerAssetContract.allowance(
  wallet.address,
  limitOrderContractAddress,
);

if (currentAllowance < makingAmount) {
  // Approve just the necessary amount or the full MaxUint256 to avoid repeated approvals
  const approveTx = await makerAssetContract.approve(
    limitOrderContractAddress,
    makingAmount,
  );
```

```
    await approveTx.wait();
}
```

Additional notes:

- ERC-20 allowances are token- and spender-specific: if you change either the token or the contract, a new approval is needed.
- To simplify future orders, you can approve a large amount once:

```
await makerAssetContract.approve(limitOrderContractAddress, MaxUint256);
```

Step 4: Define order parameters and traits

The `makerTraits` define how your order can be filled. This includes rules for partial fills, expiration time, and whether the order can be used more than once. Traits are encoded as a uint256 bitmask using a builder-style API.

In this tutorial, we'll use the following methods:

- `.withExpiration(timestamp)`: sets when the order expires.
- `.allowPartialFills()`: allows the order to be filled partially.
- `.allowMultipleFills()`: allows the order to be filled more than once.

```
const makerTraits = new MakerTraits()
  .withExpiration(expiration)
  .allowPartialFills()
  .allowMultipleFills();
```

Once the traits are configured, you can create a limit order:

```
const order = new LimitOrder({
  makerAsset: new Address(makerAsset),
  takerAsset: new Address(takerAsset),
  makingAmount,
  takingAmount,
  maker: new Address(wallet.address),
  receiver: new Address(wallet.address),
  salt: BigInt(Math.floor(Math.random() * 1e8)), // must be unique for each order
  makerTraits,
});
```

Step 5: Sign the order (EIP-712)

To make the order usable on-chain or off-chain, it must be signed according to the EIP-712 standard. `signTypedData` signs a structured payload that includes the full order definition. The resulting signature proves that the order was created and authorized by the wallet owner.

```
const typedData = order.getTypedData(domain);

// Adapt domain format for signTypedData
const domainForSignature = {
  ...typedData.domain,
  chainId: chainId,
};

const signature = await wallet.signTypedData(
  domainForSignature,
  { Order: typedData.types.Order },
  typedData.message,
);
```

Step 6: Submit the signed order

To make your signed order visible to resolvers and takers, you need to submit it to the 1inch Orderbook API using the SDK's built-in `Api` class.

This process requires:

- A valid 1inch API key
- A network ID (e.g., 1 for Ethereum Mainnet)
- A connector (e.g., AxiosProviderConnector) for HTTP requests

You automatically get your API key after successfully registering at the 1inch Developer Portal. Include it in your `.env` file as `1INCH_API_KEY=your_api_key_here`.

> ⚠ CAUTION
>
> CORS warning (for frontend use)
> If you call the API directly from a browser, you'll likely encounter CORS errors. To avoid this, use a backend proxy server that forwards the order submission securely. See a FAQ article on CORS errors for more information.

```
import { AxiosProviderConnector } from "@1inch/limit-order-sdk";

const api = new Api({
  networkId: chainId, // 1 = Ethereum Mainnet
  authKey: process.env["1INCH_API_KEY"], // Load API key securely
  httpConnector: new AxiosProviderConnector(),
});

try {
  const result = await api.submitOrder(order, signature);
  console.log("Order submitted successfully:", result);
} catch (error) {
  console.error("Failed to submit order:", error);
}
```

Conclusion

You've now completed the full process of creating, signing, and submitting a Limit Order v4 using the 1inch SDK. Your order is live and can be discovered and filled by resolvers across the network.

For more detailed technical information, refer to the 1inch Limit Order Protocol documentation.

Previous
‹ Introduction

Next
Include a limit order to the 1inch limit orders database ›