Comprehensive Coinbase Developer Platform (CDP) SDK Documentation

Table of Contents

Introduction

The Coinbase Developer Platform (CDP) SDK provides a comprehensive set of tools for interacting with blockchain networks, managing wallets, and performing various crypto operations. This document serves as a detailed guide for developers looking to integrate the CDP SDK into their applications.

Installation

Install the CDP SDK using npm:

```bash
npm install @coinbase/coinbase-sdk
```

SDK Configuration

Configuring the SDK

Configure the SDK with your API key.

```typescript
import { Coinbase } from "@coinbase/coinbase-sdk";

Coinbase.configureFromJson({ filePath: '~/Downloads/cdp_api_key.json' });
```

Parameters:
- `filePath`: String path to the JSON file containing your CDP API key.

Example:
```typescript
Coinbase.configureFromJson({ filePath: '/home/user/cdp_api_key.json' });
```

Enabling Server-Signer

Enable the Server-Signer for enhanced security in production environments.

```typescript
```

```
Coinbase.useServerSigner = true;
```

Wallet Management

Creating a Wallet

Create a new wallet on a specified network.

```typescript
import { Wallet, Coinbase } from "@coinbase/coinbase-sdk";

let wallet = await Wallet.create({ networkId: Coinbase.networks.BaseMainnet });
```

Parameters:
- `networkId`: (Optional) The network ID for the wallet. Defaults to Base Sepolia testnet.
NodeJS Network Labels:

Network Name	Coinbase.networks Constant
Base Mainnet	Coinbase.networks.BaseMainnet
Base Sepolia	Coinbase.networks.BaseSepolia
Ethereum Mainnet	Coinbase.networks.EthereumMainnet
Polygon Mainnet	Coinbase.networks.PolygonMainnet
Bitcoin Mainnet	Coinbase.networks.BitcoinMainnet
Arbitrum Mainnet	Coinbase.networks.ArbitrumMainnet
Optimism Mainnet	Coinbase.networks.OptimismMainnet

Example:
```typescript
let mainnetWallet = await Wallet.create({ networkId: Coinbase.networks.BaseMainnet });
let testnetWallet = await Wallet.create(); // Defaults to Base Sepolia
```

Exporting a Wallet

Export wallet data for persistence.

```typescript
let data = wallet.export();
```

Example:
```typescript
let exportedData = wallet.export();
console.log("Exported wallet data:", exportedData);
```

Importing a Wallet

Import a previously exported wallet.

```typescript
let importedWallet = await Wallet.import(fetchedData);
```

Parameters:
- `fetchedData`: The exported wallet data object.

Example:
```typescript
let storedData = await fetchWalletDataFromStorage();
let restoredWallet = await Wallet.import(storedData);
```

Saving Wallet Seed Locally

Save the wallet seed to a local file (for development purposes only).

```typescript
wallet.saveSeed(filePath, encrypt);
```

Parameters:
- `filePath`: String path where the seed will be saved.
- `encrypt`: Boolean indicating whether to encrypt the seed.

Example:
```typescript
wallet.saveSeed('my_wallet_seed.json', true);
```

Loading Wallet Seed

Load a previously saved wallet seed.

```typescript
await wallet.loadSeed(filePath);
```

Parameters:
- `filePath`: String path to the saved seed file.

Example:
```typescript
await wallet.loadSeed('my_wallet_seed.json');
```

Address Management

Getting the Default Address

Retrieve the default address of a wallet.

```typescript
let address = await wallet.getDefaultAddress();
```

Example:
```typescript
let defaultAddress = await wallet.getDefaultAddress();
console.log("Default address:", defaultAddress.toString());
```

Creating a New Address

Create a new address within a wallet.

```typescript
let newAddress = await wallet.createAddress();
```

Example:
```typescript
let additionalAddress = await wallet.createAddress();
console.log("New address created:", additionalAddress.toString());
```

Listing Addresses

List all addresses in a wallet.

```typescript
let addresses = wallet.getAddresses();
```

Example:
```typescript
let allAddresses = wallet.getAddresses();
allAddresses.forEach(address => console.log(address.toString()));
```

Transfers

Creating a Transfer

Initiate a transfer of assets from one wallet to another.
ETH's asset ID is Coinbase.assets.Eth
USDC's asset ID is Coinbase.assets.Usdc
WETH's asset ID is Coinbase.assets.Weth

```typescript
let transfer = await wallet.createTransfer({
  amount: number,
  assetId: string,
  destination: string | Wallet,
  gasless?: boolean
});
```

Parameters:
- `amount`: Number representing the amount to transfer.
- `assetId`: String identifier of the asset to transfer (e.g., `Coinbase.assets.Eth`).
- `destination`: Destination wallet or address string.
- `gasless`: (Optional) Boolean to indicate if the transfer should be gasless (for supported assets).

Example:
```typescript
let transfer = await wallet.createTransfer({
```

```
  amount: 0.001,
  assetId: Coinbase.assets.Eth,
  destination: "0x742d35Cc6634C0532925a3b844Bc454e4438f44e"
});
await transfer.wait();
```

Checking Transfer Status

Check the status of a transfer.

```typescript
let status = await transfer.getStatus();
```

Example:
```typescript
let transferStatus = await transfer.getStatus();
console.log("Transfer status:", transferStatus);
```

Trades

Creating a Trade

Initiate a trade between two assets.

```typescript
let trade = await wallet.createTrade({
  amount: number,
  fromAssetId: string,
  toAssetId: string
});
```

Parameters:
- `amount`: Number representing the amount to trade.
- `fromAssetId`: String identifier of the asset to trade from.
- `toAssetId`: String identifier of the asset to trade to.

Example:
```typescript
let trade = await wallet.createTrade({
  amount: 0.1,
  fromAssetId: Coinbase.assets.Eth,
  toAssetId: Coinbase.assets.Usdc
});
await trade.wait();
```

Checking Trade Status

Check the status of a trade.

```typescript
let status = await trade.getStatus();
```

Example:
```typescript
let tradeStatus = await trade.getStatus();
console.log("Trade status:", tradeStatus);
```

Smart Contract Interactions

Invoking a Contract

Invoke a method on a smart contract.

```typescript
let contractInvocation = await wallet.invokeContract({
  contractAddress: string,
  method: string,
  args: object,
  abi?: object[],
  amount?: number,
  assetId?: string
});
```

Parameters:
- `contractAddress`: String address of the contract.
- `method`: String name of the method to invoke.
- `args`: Object containing method arguments.
- `abi`: (Optional) Array of objects describing the contract ABI.
- `amount`: (Optional) Number representing the amount of native asset to send with the transaction.
- `assetId`: (Optional) String identifier of the asset to send (for payable functions).

Example:
```typescript
let contractInvocation = await wallet.invokeContract({
  contractAddress: "0x742d35Cc6634C0532925a3b844Bc454e4438f44e",
  method: "transfer",
  args: {
    to: "0xRecipientAddress",
    value: "1000000000000000000" // 1 token with 18 decimals
  },
  abi: [{
    "inputs": [
      { "name": "to", "type": "address" },
      { "name": "value", "type": "uint256" }
    ],
    "name": "transfer",
    "outputs": [{ "name": "", "type": "bool" }],
    "type": "function"
  }]
});
await contractInvocation.wait();
```

Token Deployments

Deploying an ERC-20 Token

Deploy a new ERC-20 token contract.

```typescript
let erc20 = await wallet.deployToken({
  name: string,
  symbol: string,
  totalSupply: number
});
```

Parameters:
- `name`: String name of the token.
- `symbol`: String symbol of the token.
- `totalSupply`: Number representing the total supply of tokens.

Example:
```typescript
let myToken = await wallet.deployToken({
  name: "My Token",
  symbol: "MTK",
  totalSupply: 1000000
});
console.log("Token deployed at:", myToken.getContractAddress());
```

Deploying an ERC-721 Token (NFT)

Deploy a new ERC-721 token (NFT) contract.

```typescript
let nft = await wallet.deployNFT({
  name: string,
  symbol: string,
  baseURI: string
});
```

Parameters:
- `name`: String name of the NFT collection.
- `symbol`: String symbol of the NFT collection.
- `baseURI`: String base URI for token metadata.

Example:
```typescript
let myNFT = await wallet.deployNFT({
  name: "My NFT Collection",
  symbol: "MNFT",
  baseURI: "https://api.mynft.com/metadata/"
});
console.log("NFT contract deployed at:", myNFT.getContractAddress());
```

Deploying an ERC-1155 Token (Multi-Token)

Deploy a new ERC-1155 token (Multi-Token) contract.

```typescript
```

```
let multiToken = await wallet.deployMultiToken({
  uri: string
});
```

Parameters:
- `uri`: String URI for token metadata.

Example:
```typescript
let myMultiToken = await wallet.deployMultiToken({
  uri: "https://api.mymultitoken.com/metadata/{id}.json"
});
console.log("Multi-Token contract deployed at:",
myMultiToken.getContractAddress());
```

Message Signing

Signing a Message

Sign a message using EIP-191 standard.

```typescript
import { hashMessage } from "@coinbase/coinbase-sdk";

let payloadSignature = await wallet.createPayloadSignature(hashMessage(message));
```

Parameters:
- `message`: String message to be signed.

Example:
```typescript
let message = "Hello, Coinbase!";
let signature = await wallet.createPayloadSignature(hashMessage(message));
await signature.wait();
console.log("Signature:", signature.toString());
```

Signing Typed Data

Sign typed structured data using EIP-712 standard.

```typescript
import { hashTypedData } from "@coinbase/coinbase-sdk";

let payloadSignature = await wallet.createPayloadSignature(hashTypedData({
  domain: object,
  types: object,
  primaryType: string,
  message: object
}));
```

Parameters:
- `domain`: Object containing domain data.
- `types`: Object describing the structure of the data.

- `primaryType`: String name of the primary type being signed.
- `message`: Object containing the data to be signed.

Example:
```typescript
let typedData = {
  domain: {
    name: "My dApp",
    version: "1",
    chainId: 1,
    verifyingContract: "0x1234567890123456789012345678901234567890"
  },
  types: {
    Person: [
      { name: "name", type: "string" },
      { name: "wallet", type: "address" }
    ]
  },
  primaryType: "Person",
  message: {
    name: "John Doe",
    wallet: "0x0123456789012345678901234567890123456789"
  }
};

let signature = await wallet.createPayloadSignature(hashTypedData(typedData));
await signature.wait();
console.log("Typed data signature:", signature.toString());
```

Balances and Transactions

Listing Balances

List balances for all assets in a wallet.

```typescript
let balances = await wallet.listBalances();
```

Example:
```typescript
let allBalances = await wallet.listBalances();
console.log("Wallet balances:", allBalances.toString());
```

Getting Balance for Specific Asset

Get the balance of a specific asset in a wallet.

```typescript
let balance = await wallet.getBalance(assetId);
```

Parameters:
- `assetId`: String identifier of the asset.

Example:

```typescript
let ethBalance = await wallet.getBalance(Coinbase.assets.Eth);
console.log("ETH balance:", ethBalance.toString());
```

Listing Transactions

List transactions for an address.

```typescript
let transactions = await address.listTransactions(options);
```

Parameters:
- `options`: (Optional) Object containing listing options.

Example:
```typescript
let recentTransactions = await address.listTransactions({ limit: 10 });
recentTransactions.forEach(tx => console.log(tx.toString()));
```

Server-Signer Integration

Verifying Server-Signer Assignment

Verify if a Server-Signer is assigned to your CDP project.

```typescript
import { ServerSigner } from "@coinbase/coinbase-sdk";

let serverSigner = await ServerSigner.getDefault();
```

Example:
```typescript
try {
  let signer = await ServerSigner.getDefault();
  console.log("Server-Signer is assigned:", signer);
} catch (error) {
  console.error("No Server-Signer assigned:", error);
}
```

Error Handling

The CDP SDK uses custom error types for different scenarios. Always wrap your SDK calls in try-catch blocks to handle potential errors gracefully.

Example:
```typescript
import { TimeoutError } from '@coinbase/coinbase-sdk';

try {
  let transfer = await wallet.createTransfer({
    amount: 0.001,
    assetId: Coinbase.assets.Eth,
    destination: "0x742d35Cc6634C0532925a3b844Bc454e4438f44e"
```

```
  });
  await transfer.wait();
} catch (error) {
  if (error instanceof TimeoutError) {
    console.log("Transfer timed out, check status later");
  } else {
    console.error("Error during transfer:", error);
  }
}
```

Contract Reads

```
const result = await readContract({
    networkId: "base-mainnet",
    contractAddress: "0xContractAddress",
    method: "balanceOf",
    args: { account: "0xAddress" },
    abi: [/* Optional ABI array */]
});
```

This comprehensive guide covers the major functionalities of the CDP SDK. For the most up-to-date and detailed information, always refer to the official CDP SDK documentation.