unWallet

SIVIRA Inc.

Apr 19, 2022

CONTENTS

unWallet is a non-custodial contract wallet that can be used via web browsers without installing native apps or browser extensions. Developers can integrate their dapps with unWallet using EIP1193-compliant *unWallet provider*.

CHAPTER

ONE

UNWALLET PROVIDER

unWallet provider is an EIP1193-compliant provider that provides access to unWallet.

1.1 Quick Start

1.1.1 Installation

```
$ npm install unwallet-provider
```

1.1.2 Setup

```
import { UnWalletProvider } from "unwallet-provider";

const provider = new UnWalletProvider();
```

1.1.3 EIP1102-compliant account exposure

```
const accounts = await provider.request<string[]>({
  method: "eth_requestAccounts",
});
```

1.1.4 Sending RPC request

```
const txHash = await provider.request<string>({
  method: "eth_sendTransaction",
  params: [
    {
      from: "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
      to: "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
      gas: "0x76c0",
      gasPrice: "0x9184e72a000",
      value: "0x9184e72a",
      data:
↪"0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675",
```

(continues on next page)

(continued from previous page)

```
    },
  ],
});
```

1.1.5 Disconnect from wallet

```
await provider.disable();
```

1.2 Wrapping with other libraries

1.2.1 ethers.js

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

const web3Provider = new ethers.providers.Web3Provider(provider);
```

1.3 Supported RPC methods

Note: RPC methods other than the following are available by setting arbitrary RPC endpoints to the provider. See *Configuration* for details.

1.3.1 eth_requestAccounts

Parameters

none

Returns

`Array` of `DATA (20 Bytes)` - addresses that the user approved to access

Example

```
// Request
const accounts = await provider.request<string[]>({
  method: "eth_requestAccounts",
});

// Result
["0x407d73d8a49eeb85d32cf465507dd71d507100c1"]
```

1.3.2 eth_accounts

Parameters

none

Returns

`Array` of `DATA (20 Bytes)` - addresses that the user approved to access

Example

```
// Request
const accounts = await provider.request<string[]>({
  method: "eth_accounts",
});

// Result
["0x407d73d8a49eeb85d32cf465507dd71d507100c1"]
```

1.3.3 eth_chainId

Parameters

none

Returns

`Number` - integer of the chain ID currently connected

Example

```
// Request
const chainId = await provider.request<number>({
  method: "eth_chainId",
});

// Result
1
```

1.3.4 personal_sign

Parameters

1. `DATA` - message to be signed
2. `DATA (20 Bytes)` - address of the account that will sign the message

Returns

`DATA` - signature

Example

```
// Request
const sig = await provider.request<string>({
  method: "personal_sign",
  params: [
    "0xdeadbeaf",
    "0x9b2055d370f73ec7d8a03e965129118dc8f5bf83",
  ],
});

// Result

↪"0xa3f20717a250c2b0b729b7e5becbff67fdaef7e0699da4de7ca5895b02a170a12d887fd3b17bfdce3481f10bea41f45ba9
↪"
```

1.3.5 eth_sign

Parameters

1. `DATA (20 Bytes)` - address of the account that will sign the message
2. `DATA` - message to be signed

Returns

`DATA` - signature

Example

```
// Request
const sig = await provider.request<string>({
  method: "eth_sign",
  params: [
    "0x9b2055d370f73ec7d8a03e965129118dc8f5bf83",
    "0xdeadbeaf",
  ],
```

(continues on next page)

(continued from previous page)

```
});

// Result
↪"0xa3f20717a250c2b0b729b7e5becbff67fdaef7e0699da4de7ca5895b02a170a12d887fd3b17bfdce3481f10bea41f45ba9
↪"
```

1.3.6 eth_signTypedData

Parameters

1. `DATA (20 Bytes)` - address of the account that will sign the messages
2. `Object` - EIP712-compliant typed structured data to be signed

Returns

`DATA` - signature

Example

```
// Request
const sig = await provider.request<string>({
  method: "eth_signTypedData",
  params: [
    "0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826",
    {
      types: {
        EIP712Domain: [
          {
            name: "name",
            type: "string",
          },
          {
            name: "version",
            type: "string",
          },
          {
            name: "chainId",
            type: "uint256",
          },
          {
            name: "verifyingContract",
            type: "address",
          },
        ],
        Person: [
          {
            name: "name",
```

(continues on next page)

(continued from previous page)

```
            type: "string",
          },
          {
            name: "wallet",
            type: "address",
          },
        ],
        Mail: [
          {
            name: "from",
            type: "Person",
          },
          {
            name: "to",
            type: "Person",
          },
          {
            name: "contents",
            type: "string",
          },
        ],
      },
      primaryType: "Mail",
      domain: {
        name: "Ether Mail",
        version: "1",
        chainId: 1,
        verifyingContract: "0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCcccccccC",
      },
      message: {
        from: {
          name: "Cow",
          wallet: "0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826",
        },
        to: {
          name: "Bob",
          wallet: "0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB",
        },
        contents: "Hello, Bob!",
      },
    },
  ],
});

// Returns
↪"0x4355c47d63924e8a72e509b65029052eb6c299d53a04e167c5775fd466751c9d07299936d304c153f6443dfa05f40ff007
↪"
```

1.3.7 eth_signTypedData_v4

Note: This method is provided for compatibility with MetaMask.

Parameters

1. `DATA (20 Bytes)` - address of the account that will sign the messages
2. `String` - JSON encoded EIP712-compliant typed structured data to be signed

Returns

`DATA` - signature

Example

```
// Request
const sig = await provider.request<string>({
  method: "eth_signTypedData_v4",
  params: [
    "0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826",
    `{"types":{"EIP712Domain":[{"name":"name","type":"string"},{"name":"version","type":
↪"string"},{"name":"chainId","type":"uint256"},{"name":"verifyingContract","type":
↪"address"}],"Person":[{"name":"name","type":"string"},{"name":"wallet","type":"address
↪"}],"Mail":[{"name":"from","type":"Person"},{"name":"to","type":"Person"},{"name":
↪"contents","type":"string"}]},"primaryType":"Mail","domain":{"name":"Ether Mail",
↪"version":"1","chainId":1,"verifyingContract":
↪"0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCcccccccC"},"message":{"from":{"name":"Cow","wallet":
↪"0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826"},"to":{"name":"Bob","wallet":
↪"0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB"},"contents":"Hello, Bob!"}}`,
  ],
});

// Returns

↪"0x4355c47d63924e8a72e509b65029052eb6c299d53a04e167c5775fd466751c9d07299936d304c153f6443dfa05f40ff007
↪"
```

1.3.8 eth_sendTransaction

Parameters

1. `Object` - transaction object

- `from`: `DATA (20 Bytes)` - (optional) address that the transaction is send from
- `to`: `DATA (20 Bytes)` - address that the transaction is directed to
- `gas`: `QUANTITY` - (optional) integer of the gas provided for the transaction execution

- `gasPrice`: `QUANTITY` - (optional) integer of the gas price used for each paid gas
- `value`: `QUANTITY` - (optional) integer of the value sent with the transaction
- `data`: `DATA` - (optional) hash of the invoked method signature and encoded parameters

Returns

`DATA (32 Bytes)` - transaction hash

Example

```
const txHash = await provider.request<string>({
  method: "eth_sendTransaction",
  params: [
    {
      from: "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
      to: "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
      gas: "0x76c0",
      gasPrice: "0x9184e72a000",
      value: "0x9184e72a",
      data:
↪"0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675",
    },
  ],
});
```

1.3.9 wallet_switchEthereumChain

Note: See also EIP3326.

1. `Object`

- `chainId`: integer ID of the chain as a hexadecimal string

Returns

`null`

Example

```
await provider.request<null>({
  method: "wallet_switchEthereumChain",
  params: [
    {
      chainId: "0x1",
    },
  ],
});
```

1.4 Configuration

1.4.1 rpc

You can execute RPC methods other than *Supported RPC methods* by setting arbitrary RPC endpoints to the provider as follows.

```
const provider = new UnWalletProvider({
  rpc: {
    // <CHAIN_ID>: "<ENDPOINT>",
    1: "https://mainnet.infura.io/v3/YOUR_PROJECT_ID",
    137: "https://polygon-mainnet.infura.io/v3/YOUR_PROJECT_ID",
  },
});

const count = await provider.request<string>({
  method: "eth_getTransactionCount",
  params: [
    "0x407d73d8a49eeb85d32cf465507dd71d507100c1",
  ],
});
```

1.4.2 allowAccountsCaching

If `allowAccountsCaching` option is `true`, the provider caches information about the accounts in local storage so that you do not have to execute eth_requestAccounts each time you instantiate the provider.

```
const provider = new UnWalletProvider({
  allowAccountsCaching: true,
});
```