
plasma-js-lib Documentation

Release latest

Mar 08, 2019

1	Getting Started	3
1.1	Adding plasma-js-lib	3
2	Contributing	5
2.1	Contributing Guide and Code of Conduct	5
2.2	Getting Started as a Contributor	5
3	PlasmaClient	7
3.1	getAccounts	7
3.2	getBalances	8
3.3	getExits	8
3.4	getTransaction	9
3.5	getBlock	10
3.6	getCurrentBlock	10
3.7	getNextBlock	10
3.8	getTokenId	11
3.9	createAccount	11
3.10	sign	12
3.11	deposit	12
3.12	pickRanges	14
3.13	sendRawTransaction	14
3.14	sendTransaction	15
3.15	startExit	16
3.16	finalizeExits	16
3.17	listToken	17
4	PlasmaOperator	19
4.1	getBlockMetadata	19
4.2	getBlockTransactions	20
4.3	getTransaction	20
4.4	getRecentTransactions	21
4.5	getCurrentBlock	22
4.6	submitBlock	22

Hello and welcome to the documentation of Plasma Group's `plasma-js-lib`! `plasma-js-lib` is a simple JavaScript library that makes it easy for you to interact with a plasma chain.

Hello! If you're looking to build your first plasma chain application, you're in the right place.

`plasma-js-lib` is a JavaScript library that makes it easy for you to interact with plasma chains. This includes things like making transactions, querying balances, querying blocks, and a lot more.

1.1 Adding plasma-js-lib

There are a few simple ways to add `plasma-js-lib` to your project.

1.1.1 npm

If you're working with a project that supports `npm` imports, you can install `plasma-js-lib` with `npm`:

Then you'll be able to import `Plasma` in your project:

1.1.2 Browser

You can also import `plasma-js-lib` with a `<script>` tag:

This will give you access to a window variable:

CHAPTER 2

Contributing

Welcome! A huge thank you for your interest in contributing to Plasma Group. Plasma Group is an open source initiative developing a simple and well designed [plasma](#) implementation. If you're looking to contribute to `plasma-js-lib`, you're in the right place! It's contributors like you that make open source projects work, we really couldn't do it without you.

We don't just need people who can contribute code. We need people who can run this code for themselves and break it. We need people who can report bugs, request new features, and leave helpful comments. **We need you!**

We're always available to answer your questions and to help you become a contributor! You can reach out to any of the [members of Plasma Group](#) on GitHub, or send us an email at contributing@plasma.group.

Here at Plasma Group we're trying to foster an inclusive, welcoming, and accessible open source ecosystem. The best open source projects are those that make contributing an easy and rewarding experience. We're trying to follow those best practices by maintaining a series of resources for contributors to Plasma Group repositories.

If you're a new contributor to `plasma-js-lib`, please read through the following information. These resources will help you get started and will help you better understand what we're building.

2.1 Contributing Guide and Code of Conduct

Plasma Group follows a [Contributing Guide and Code of Conduct](#) adapted slightly from the [Contributor Covenant](#). **All contributors are expected to read through this guide.** We're here to cultivate a welcoming and inclusive contributing environment. Every new contributor needs to do their part to uphold our community standards.

2.2 Getting Started as a Contributor

2.2.1 Requirements and Setup

Cloning the Repository

First, clone the `plasma-js-lib` GitHub repository and enter the directory:

Node.js

`plasma-js-lib` is a [Node.js](#) application. You'll need to install `Node.js` (and its corresponding package manager, `npm`) for your system before continuing.

Packages

`plasma-js-lib` makes use of several `npm` packages.

Install all required packages with:

2.2.2 Running Tests

`plasma-js-lib` makes use of a combination of [Mocha](#) (a testing framework) and [Chai](#) (an assertion library) for testing.

Run all tests with:

Contributors: remember to run tests before submitting a pull request! Code with passing tests makes life easier for everyone and means your contribution can get pulled into this project faster.

CHAPTER 3

PlasmaClient

PlasmaClient handles interaction with plasma clients that implement the PG JSON-RPC Calls

```
const PlasmaClient = require('plasma-js-lib')

// Connects automatically to http://localhost:9898
const plasma = new PlasmaClient()
```

3.1 getAccounts

```
plasma.getAccounts()
```

Returns the list of available accounts.

3.1.1 Returns

Promise<Array>: List of addresses controlled by the node.

3.1.2 Example

```
const accounts = await plasma.getAccounts()
console.log(accounts)
> [ '0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6',
    '0x946E85B7C74a89f2710BEa6Cb83d4c1AEA40682F',
    '0xbF699b0d6e59B865d74D9D1714A407f6516B0F60' ]
```

3.2 getBalances

```
plasma.getBalances(address)
```

Returns all token balances for an address. Balances are returned as BigNum.

3.2.1 Parameters

1. address - string: Address to return balances for.

3.2.2 Returns

Promise<Object>: A mapping of token IDs to account balances.

3.2.3 Example

```
const balances = await plasma.getBalances('0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6')
console.log(balances)
> { '0': <BN: 64> }
```

3.3 getExits

```
plasma.getExits(address)
```

Returns all active exits for an address.

3.3.1 Parameters

1. address - string: Address to return exits for.

3.3.2 Returns

Promise<Array>: List of exits.

3.3.3 Example

```
const exits = await plasma.getExits('0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6')
console.log(exits)
> [ { token: '0',
      start: '384',
      end: '3e8',
      id: '0',
      block: '3a5b57',
```

(continues on next page)

(continued from previous page)

```

    exiter: '0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6',
    completed: true,
    finalized: false } ]

```

3.4 getTransaction

```
plasma.getTransaction(hash)
```

Returns a transaction given its hash.

3.4.1 Parameters

1. hash - string: Hash of the transaction to return.

3.4.2 Returns

Promise<SignedTransaction>: Transaction with the given hash.

3.4.3 Example

```

const transaction = await plasma.getTransaction(
  ↪ '0xae5ac607d29c6d38a63db00550160b5ca3b51ec9b3ede8dcb5755b60700aecfe')
console.log(transaction)
> SignedTransaction {
  schema:
    Schema {
      unparsedFields:
        { block: [Object], transfers: [Object], signatures: [Object] },
      fields:
        { block: [SchemaNumber],
          transfers: [Schema],
          signatures: [Schema] } },
  block: <BN: 389e>,
  transfers:
    [ { sender: '0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6',
      recipient: '0x946E85B7C74a89f2710BEa6Cb83d4c1AEA40682F',
      token: <BN: 0>,
      start: <BN: 3e8>,
      end: <BN: 44c> } ],
  signatures:
    [ { v: <Buffer 1b>,
      r: <Buffer 07 78 c7 ba a3 df 5e 4d 39 ff 4a 17 63 f5 53 84 4a 30 b5 47 1a_
↪ 75 71 06 f7 a5 f7 e2 f7 00 91 1b>,
      s: <Buffer 53 94 5b 03 2d fb a3 4d 9f 59 13 a1 06 ff 09 0e 88 b6 19 4b 27_
↪ 74 9a c4 e9 31 17 2c 0c b7 6e d1> } ] }

```

3.5 getBlock

```
plasma.getBlock(block)
```

Returns the hash of the plasma block with the given number.

3.5.1 Parameters

1. `block - number`: Number of the block to query.

3.5.2 Returns

`Promise<string>`: Hash of the block with that number.

3.5.3 Example

```
const block = await plasma.getBlock(15)
console.log(block)
> 0x26e5955d5db3d1fb3fd4142fbf91daa9d8f6b58f0612c6e52eee79ca7755b004
```

3.6 getCurrentBlock

```
plasma.getCurrentBlock()
```

Returns the number of the most recently submitted block.

3.6.1 Returns

`Promise<number>`: Last submitted block number.

3.6.2 Example

```
const currentBlock = await plasma.getCurrentBlock()
console.log(currentBlock)
> 5442
```

3.7 getNextBlock

```
plasma.getNextBlock()
```

Returns the number of the plasma block that will be submitted next.

3.7.1 Returns

Promise<number>: Next plasma block number.

3.7.2 Example

```
const nextBlock = await plasma.getNextBlock()
console.log(nextBlock)
> 5443
```

3.8 getTokenId

```
plasma.getTokenId(tokenAddress)
```

Returns the **token ID** of the token at the given contract address.

3.8.1 Parameters

1. tokenAddress - string: Address of the contract that represents the token.

3.8.2 Returns

Promise<string>: The token's ID.

3.8.3 Example

```
const tokenId = await plasma.getTokenId('0xf88ce35b57e37cda8a8520f1a290b7edef532d95')
console.log(tokenId)
> 1
```

3.9 createAccount

```
plasma.createAccount()
```

Creates a new account.

3.9.1 Returns

Promise<string>: Address of the created account.

3.9.2 Example

```
const account = await plasma.createAccount()
console.log(account)
> 0x8508c8aCA521512D4695eCF6976d2e8D2666a46d
```

3.10 sign

```
plasma.sign(address, data)
```

Signs a message with a given account.

3.10.1 Parameters

1. address - string: Address of the account to sign with.
2. data - string: Message to sign.

3.10.2 Returns

Promise<Object>: An [Ethereum signature object](#).

3.10.3 Example

```
const signature = await plasma.sign('0x8508c8aCA521512D4695eCF6976d2e8D2666a46d',
  ↪ 'Hello!')
console.log(signature)
> { message: 'Hello!',
  messageHash: '0x52b6437db56d87f5991d7c173cf11b9dd0f9fb083260bef1bf0c338042bc398c',
  v: '0x1c',
  r: '0x47de6cc9f808658d643c3fd4a79be725627f719e6604d86f7b6356f3bdb81ed3',
  s: '0x4e18918c4b0a60dfa2ce3ee623c815b90b4eb30f5a83bae5b89778ff0aa742af',
  signature:
  ↪ '0x47de6cc9f808658d643c3fd4a79be725627f719e6604d86f7b6356f3bdb81ed34e18918c4b0a60dfa2ce3ee623c815b'
  ↪ }
```

3.11 deposit

```
plasma.deposit(token, amount, address)
```

Deposits an amount of a given token for an address.

(continued from previous page)

```
id: 'log_a5079148',
returnValues: [Object],
event: 'DepositEvent',
signature:
→ '0x7a9ec4e041f302c44606a6b6c9f3ab369e99b054e8582f4fc4d6f39240cfc810',
raw: [Object] } } }
```

3.12 pickRanges

```
plasma.pickRanges(address, token, amount)
```

Picks the best ranges to make a transaction.

3.12.1 Parameters

1. address - string: Address to transact from.
2. token - string: ID or address of token to send.
3. amount - number: Amount to be sent.

3.12.2 Returns

Promise<Array>: An array of [Range](#) objects.

3.12.3 Example

```
const ranges = await plasma.pickRanges('0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6',
→ '1', 5000)
console.log(ranges)
> [ { token: '1',
    start: '0',
    end: '1388',
    owner: '0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6' } ]
```

3.13 sendRawTransaction

```
plasma.sendRawTransaction(transaction)
```

Sends an encoded and signed transaction to the operator. If you're looking for an easier way to send transactions, look at `sendTransaction` below.

3.13.1 Parameters

1. `transaction - string`: The encoded signed transaction.

3.13.2 Returns

`Promise<string>`: A transaction receipt.

3.13.3 Example

```
const receipt = await plasma.sendRawTransaction(
  ↳ '0000389e011E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6946E85B7C74a89f2710BEa6Cb83d4c1AEA40682F00000000'
  ↳ ')
console.log(receipt)
>
↳ 0000389e011E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6946E85B7C74a89f2710BEa6Cb83d4c1AEA40682F00000000
```

3.14 sendTransaction

```
plasma.sendTransaction(from, to, token, amount)
```

The method that most people should use to make transactions. Wraps `sendRawTransaction` and automatically calculates the best ranges for a given transaction. Also handles formatting and signing the transaction.

3.14.1 Parameters

1. `from - string`: Address to send from.
2. `to - string`: Address to send to.
3. `token - string`: ID or address of the token to send.
4. `amount - number`: Amount of the token to send.

3.14.2 Returns

`Promise<string>`: A transaction receipt.

3.14.3 Example

```
const receipt = await plasma.sendTransaction(
  ↳ '0x82A978B3f5962A5b0957d9ee9eEf472EE55B42F1',
  ↳ '0x7d577a597B2742b498Cb5Cf0C26cDCD726d39E6e', '0', 50)
console.log(receipt)
>
↳ 000000030182A978B3f5962A5b0957d9ee9eEf472EE55B42F17d577a597B2742b498Cb5Cf0C26cDCD726d39E6e00000000
```

3.15 startExit

```
plasma.startExit(address, token, amount)
```

Starts exits for a user to withdraw a certain amount of a given token. Will automatically select the right ranges to withdraw and submit more than one exit if necessary.

3.15.1 Parameters

1. `address - string`: Address to submit exits for.
2. `token - string`: ID or address of the token to exit.
3. `amount - number`: Amount of the token to withdraw.

3.15.2 Returns

`Promise<Array>`: Ethereum transaction hash for each exit.

3.15.3 Example

```
const exitTxs = await plasma.startExit('0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6',  
  ↪ '1', 5000)  
console.log(exitTxs)  
> [ '0xfdb32a468624233010d9648e8231327d0ff9794cc8b722c2a8539d90cb5af20c' ]
```

3.16 finalizeExits

```
plasma.finalizeExits(address)
```

Finalizes all available exits for an address. Will not finalize any exits that are still in their challenge period or have already been finalized.

3.16.1 Parameters

1. `address - string`: Address to finalize exits for.

3.16.2 Returns

`Promise<Array>`: Ethereum transaction hash for each finalization.

3.16.3 Example

```
plasma.listToken(tokenAddress)
```

```
const listingTx = await plasma.listToken('0xf88ce35b57e37cda8a8520f1a290b7edef532d95')  
console.log(listingTx)  
  
> { blockHash: '0x114e62f5e92e50ed941f5ad0d63f04ad90d9677613a4897bdbbc5a6f5d3774700',  
    blockNumber: 3824586,  
    contractAddress: null,  
    cumulativeGasUsed: 676722,  
    from: '0x1e3a4a2edec2b3568b5ad0656ec3b48d9c699db6',  
    gasUsed: 92449,  
    logsBloom:  
      ↪ '0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000'  
      ↪ ,  
    status: '0x1',  
    to: '0x888c238f821fd7e62460f029adbcb388aa3143932',  
    transactionHash:  
      ↪ '0x35c88e3b51109dd446319318a3be285d065943203ebb8780bb1f9717f72a051d',  
    transactionIndex: 6,  
    events:  
      { ListingEvent:  
        { address: '0x888C238F821FD7E62460F029ADBcb388AA3143932',  
          blockHash:  
            ↪ '0x114e62f5e92e50ed941f5ad0d63f04ad90d9677613a4897bdbbc5a6f5d3774700',  
          blockNumber: 3824586,  
          logIndex: 8,  
          removed: false,  
          transactionHash:  
            ↪ '0x35c88e3b51109dd446319318a3be285d065943203ebb8780bb1f9717f72a051d',  
          transactionIndex: 6,  
          id: 'log_53a1c942',
```

17

(continued from previous page)

```
returnValues: [Object],
event: 'ListingEvent',
signature:
↪ '0x80ed85783ee3285a2a09339e1e9f1c0b2a3aa05240c97e1a741ac6347a2aca11',
raw: [Object] } } }
```

CHAPTER 4

PlasmaOperator

PlasmaOperator handles interaction with the plasma operator.

```
const PlasmaOperator = require('plasma-js-lib').PlasmaOperator

// Can replace the endpoint with the endpoint of your operator.
const operator = new PlasmaOperator('http://localhost:3000')
```

4.1getBlockMetadata

```
operator.getBlockMetadata(start, end)
```

Returns metadata about a list of blocks.

4.1.1 Parameters

1. start - number: First block to query.
2. end - number: Last block to query.

4.1.2 Returns

Promise<Array>: A list of metadata objects for each block.

4.1.3 Example

```
const blocks = await operator.getBlockMetadata(0, 3)
console.log(blocks)
> [ { blockNumber: '00000001',
    rootHash: '0000000000000000000000000000000000000000000000000000000000000000',
    timestamp: '0168c59834e8',
    numTxs: '00' },
  { blockNumber: '00000002',
    rootHash: '0000000000000000000000000000000000000000000000000000000000000000',
    timestamp: '0168c5983bbe',
    numTxs: '00' },
  { blockNumber: '00000003',
    rootHash: '0000000000000000000000000000000000000000000000000000000000000000',
    timestamp: '0168c5983fcf',
    numTxs: '00' } ]
```

4.2 getBlockTransactions

```
operator.getBlockTransactions(block, start, end)
```

Returns the transactions in a specific block. Queries all transactions between `start` and `end`. Limited to 25 transactions at a time.

4.2.1 Parameters

1. `block - number`: Number of the block to query.
2. `start - number`: First transaction to query.
3. `end - number`: Last transaction to query.

4.2.2 Returns

`Promise<Array>`: A list of transaction objects.

4.3 getTransaction

```
operator.getTransaction(hash)
```

Returns a transaction by its hash.

4.3.1 Parameters

1. `hash - string`: Hash of the transaction to return.

4.3.2 Returns

Promise<SignedTransaction>: The transaction object.

4.3.3 Example

```
const transaction = await operator.getTransaction(
  ↪ '0x7b6ced8ecd267f504f86b6cace13f078f936a20adc98b37fc83e1030f976e8e5')
console.log(transaction)
> SignedTransaction {
  schema:
    Schema {
      unparsedFields:
        { block: [Object], transfers: [Object], signatures: [Object] },
      fields:
        { block: [SchemaNumber],
          transfers: [Schema],
          signatures: [Schema] } },
  block: <BN: 2>,
  transfers:
    [ { sender: '0x82A978B3f5962A5b0957d9ee9eEf472EE55B42F1',
      recipient: '0x7d577a597B2742b498Cb5Cf0C26cDCD726d39E6e',
      token: <BN: 0>,
      start: <BN: 0>,
      end: <BN: a> } ],
  signatures: [] }
```

4.4 getRecentTransactions

```
operator.getRecentTransactions(start, end)
```

Returns a list of recent transactions.

4.4.1 Parameters

1. start - number: First transaction to query.
2. end - number: Last transaction to query.

4.4.2 Returns

Promise<Array>: A list of transaction objects.

4.4.3 Example

```
const transactions = await operator.getRecentTransactions(0, 10)
console.log(transactions)
> [ SignedTransaction {
  schema: Schema { unparsedFields: [Object], fields: [Object] },
  block: <BN: 5>,
  transfers: [ [Object] ],
  signatures: [ [Object] ] },
  SignedTransaction {
    schema: Schema { unparsedFields: [Object], fields: [Object] },
    block: <BN: 5>,
    transfers: [ [Object] ],
    signatures: [ [Object] ] },
  SignedTransaction {
    schema: Schema { unparsedFields: [Object], fields: [Object] },
    block: <BN: 2>,
    transfers: [ [Object] ],
    signatures: [ [Object] ] } ]
```

4.5 getCurrentBlock

```
operator.getCurrentBlock()
```

Returns the current block number according to the operator.

4.5.1 Returns

Promise<number>: Current block number.

4.5.2 Example

```
const currentBlock = await operator.getCurrentBlock()
console.log(currentBlock)
> 6
```

4.6 submitBlock

```
operator.submitBlock()
```

Attempts to force the operator to submit a block. If the operator is properly configured, it won't let you do this. Usually used for testing locally.

4.6.1 Example

```
const submittedBlock = await operator.submitBlock()
console.log(submittedBlock)
> 7
```