@pigi/utils Documentation

Plasma Group

Jul 01, 2020

API Reference

Hello and welcome to the documentation of Plasma Group's `plasma-utils`! `plasma-utils` is the set of core utilities used in the rest of the Plasma Group repositories.

CHAPTER 1

Utilities

`plasma-utils.utils` provides miscellaneous utilities often used when interacting with plasma chains.

```
const utils = require('plasma-utils').utils
```

1.1 int32ToHex

```
utils.int32ToHex(x)
```

Converts a 32 byte integer to a hex string.

1.1.1 Parameters

1. `x` - `number`: A 32 byte integer.

1.1.2 Returns

`string`: The integer represented as a hex string.

1.2 sleep

```
utils.sleep(ms)
```

Sleeps for a number of milliseconds.

1.2.1 Parameters

1. `ms` - `number`: Number of milliseconds to sleep.

1.2.2 Returns

`Promise`: A promise that resolves after the given number of ms.

1.3 remove0x

```
utils.remove0x(str)
```

Removes "0x" from the start of a string, if it's present.

1.3.1 Parameters

1. `str` - `string`: String to modify.

1.3.2 Returns

`string`: The string without a leading "0x".

1.4 add0x

```
utils.add0x(x)
```

Adds "0x" to the start of a string, if not already present.

1.4.1 Parameters

1. `str` - `string`: String to modify.

1.4.2 Returns

`string`: The string with "0x".

1.5 isString

```
utils.isString(str)
```

Checks if the input value is a string.

1.5.1 Parameters

1. `str` - `any`: The thing that might be a string.

1.5.2 Returns

`boolean`: `true` if the input is a string, `false` otherwise.

1.6 getRandomElement

```
utils.getRandomElement(arr)
```

Returns a random element from an array.

1.6.1 Parameters

1. `arr` - `Array`: An array.

1.6.2 Returns

`any`: A random element from that array.

1.7 getRandomAccount

```
utils.getRandomAccount()
```

Returns a random Ethereum account.

1.7.1 Returns

`any`: The Ethereum account.

1.8 sign

```
utils.sign(data, key)
```

Signs a message with a private key.

1.8.1 Parameters

1. `data` - `string`: Message to sign.
2. `key` - `string`: Private key to sign with.

1.8.2 Returns

`string`: The signature.

1.9 signatureToString

```
utils.signatureToString(signature)
```

Converts a signature with v,r,s Buffers to a single hex string.

1.9.1 Parameters

1. `signature` - `Object`: A signature object.

1.9.2 Returns

`string`: The signature as a hex string.

1.10 stringToSignature

```
utils.stringToSignature(signature)
```

Converts a hex string signature into an object with v,r,s Buffers.

1.10.1 Parameters

1. `signature` - `string`: A signature string.

1.10.2 Returns

`Object`: A signature object with v,r,s as Buffers.

1.11 getSequentialTxs

```
utils.getSequentialTxs(n, blockNum)
```

Generates sequential transactions in the same block. Usually used for testing with mass amounts of transactions.

1.11.1 Parameters

1. `n` - `number`: Number of transactions to generate.
2. `blockNum` - `number`: Block in which the transactions will be included.

1.11.2 Returns

`Array<SignedTransaction>`: An array of SignedTransaction objects.

1.12 getRandomTx

```
utils.getRandomTx(blockNum, sender, recipient, numTransfers)
```

Generates a random transaction. Usually used for testing.

1.12.1 Parameters

1. `blockNum` - `number`: Block in which this transaction will be included.
2. `sender` - `string`: Address of the sender.
3. `recipient` - `string`: Address of the recipient.
4. `numTransfers` - `number`: Number of transfers to generate.

1.12.2 Returns

`UnsignedTransaction`: An UnsignedTransaction object.

CHAPTER 2

Logging

`plasma-utils.logging` exposes a few logging helpers. These helpers make it possible to log messages in different contexts, e.g. in development or production.

2.1 API

2.1.1 ConsoleLogger

Creates a new *ConsoleLogger*, which simply wraps *console.log*.

Parameters

N/A

Returns

`ConsoleLogger`: The `ConsoleLogger` instance.

= Methods =

Logs a new message to the console.

Parameters

1. `message` - `String`: The message to be logged.

2.1.2 DebugLogger

Creates a new *DebugLogger*, which wraps the debug NPM library. The *DebugLogger* generally has better formatting than the *ConsoleLogger*.

Parameters

N/A

Returns

`DebugLogger`: The `DebugLogger` instance.

= Methods =

Logs a new message to the console, with extra formatting.

Parameters

1. `message` - `String`: The message to be logged.

CHAPTER 3

Serialization

`plasma-utils.serialization` provides utilities for encoding and decoding transactions. It also provides tools for defining your own custom schemas using our encoding scheme.

3.1 TransferSchema

```
const TransferSchema = new Schema({
  token: {
    type: ByteArray,
    length: 4,
    required: true
  },
  start: {
    type: ByteArray,
    length: 12,
    required: true
  },
  end: {
    type: ByteArray,
    length: 12,
    required: true
  },
  sender: {
    type: Address,
    required: true
  },
  recipient: {
    type: Address,
    required: true
  }
})
```

A *Transfer* represents a portion of a transaction. Each transaction is composed of one or more *Transfer*. By allowing a transaction to support more than one transfer, we enable atomic swaps.

3.2 SignatureSchema

```
const SignatureSchema = new Schema({
  v: {
    type: ByteArray,
    length: 1,
    required: true
  },
  r: {
    type: ByteArray,
    length: 32,
    required: true
  },
  s: {
    type: ByteArray,
    length: 32,
    required: true
  }
})
```

A *Signature* is a simple representation of an ECDSA signature.

3.3 UnsignedTransactionSchema

```
const UnsignedTransactionSchema = new Schema({
  block: {
    type: Number,
    length: 4,
    required: true
  },
  transfers: {
    type: [TransferSchema]
  }
})
```

An *UnsignedTransactionSchema* is composed of one or more *Transfer* objects.

3.4 SignedTransactionSchema

```
const SignedTransactionSchema = new Schema({
  block: {
    type: Number,
    length: 4,
    required: true
  },
  transfers: {
    type: [TransferSchema]
  },
  signatures: {
    type: [SignatureSchema]
  }
})
```

A *SignedTransactionSchema* is composed of one or more *Transfer* objects and a *Signature* for each *Transfer*.

CHAPTER 4

Models

`plasma-utils` provides several different "models" that represent the various common data structures we use. These models implement the schemas that we use in serialization.

It's pretty simple to import all of the available models:

```
const utils = require('plasma-utils')
const models = utils.serialization.models
```

4.1 Transfer

A `Transfer` is the basic component of every transaction. Every transaction has one or more transfers.

```
const Transfer = models.Transfer

const transfer = new Transfer({
  token: 0,
  start: 0,
  end: 100,
  sender: '0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6',
  recipient: '0x8508c8aCA521512D4695eCF6976d2e8D2666a46d'
})
```

4.1.1 Properties

token

```
transfer.token
```

ID of the token being transferred.

Returns

`BigNum`: The token ID.

---

start

```
transfer.start
```

Start of the range being transferred.

Returns

`BigNum`: The range start.

---

typedStart

```
transfer.typedStart
```

The "typed" start of the range being transferred. Calculated by concatenating `token` and `start`. Primarily used for calculating state updates in `plasma-core`.

Returns

`BigNum`: The typed start.

---

end

```
transfer.end
```

End of the range being transferred.

Returns

`BigNum`: The range end.

---

typedEnd

```
transfer.typedEnd
```

The "typed" end of the range being transferred. Calculated by concatenating `token` and `end`. Primarily used for calculating state updates in `plasma-core`.

Returns

`BigNum`: The typed end.

---

sender

```
transfer.sender
```

Address of the user sending the transfer.

Returns

`string`: Sender address.

---

recipient

```
transfer.recipient
```

Address of the user receiving the transfer.

Returns

`string`: Recipient address.

---

encoded

```
transfer.encoded
```

The encoded version of the transfer according to the rules in our schemas.

Returns

`string`: The encoded transfer.

4.2 UnsignedTransaction

An `UnsignedTransaction` contains transfers and a block number, but no signatures.

```
const UnsignedTransaction = models.UnsignedTransaction

const unsigned = new UnsignedTransaction({
  block: 123,
  transfers: [
    {
      token: 0,
      start: 0,
      end: 100,
      sender: '0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6',
      recipient: '0x8508c8aCA521512D4695eCF6976d2e8D2666a46d'
    }
  ]
})
```

4.2.1 Properties

block

```
unsigned.block
```

The block in which this transaction was included.

Returns

`BigNum`: The transaction block number.

transfers

```
unsigned.transfers
```

A list of `Transfers` that make up this transaction.

Returns

`Array<Transfer>`: A list of transfers.

encoded

```
unsigned.encoded
```

The hex-encoded version of this transaction.

Returns

`string`: Encoded transaction.

---

hash

```
unsigned.hash
```

The keccak256 (Ethereum's SHA3) hash of the encoded transaction.

Returns

`string`: Hash of the transaction.

4.3 SignedTransaction

An `SignedTransaction` contains transfers, and a block number, and a signature for each transfer.

```
const SignedTransaction = models.SignedTransaction

const signed = new SignedTransaction({
  block: 123,
  transfers: [
    {
      token: 0,
      start: 0,
      end: 100,
      sender: '0x1E3a4a2edec2b3568B5Ad0656ec3b48d9C699dB6',
      recipient: '0x8508c8aCA521512D4695eCF6976d2e8D2666a46d'
    }
  ],
  signatures: [
    {
      v: '0x1b',
      r: '0xd693b532a80fed6392b428604171fb32fdbf953728a3a7ecc7d4062b1652c042',
      s: '0x24e9c602ac800b983b035700a14b23f78a253ab762deab5dc27e3555a750b354'
    }
  ]
})
```

4.3.1 Properties

block

```
signed.block
```

The block in which this transaction was included.

Returns

`BigNum`: The transaction block number.

---

transfers

```
signed.transfers
```

A list of `Transfers` that make up this transaction.

Returns

`Array<Transfer>`: A list of transfers.

---

signatures

```
signed.signatures
```

A list of `Signatures` on this transaction. There should be one signature for each transfer, where the signature is from the sender of the transfer.

Returns

`Array<Signature>`: A list of signatures.

---

encoded

```
signed.encoded
```

The hex-encoded version of this transaction.

Returns

`string`: Encoded transaction.

---

hash

```
signed.hash
```

The keccak256 (Ethereum's SHA3) hash of the encoded *unsigned* version of this transaction. Effectively the same as casting this transaction to an `UnsignedTransaction` and getting the hash.

Returns

`string`: Hash of the *unsigned* version of this transaction.

4.3.2 Methods

---

checkSigs

```
signed.checkSigs()
```

Checks that the signatures on the transaction are valid.

Returns

`boolean`: `true` if the transaction is valid, `false` otherwise.