Browseth Documentation

Release 0.0.59

Braden Pezeshki, Ryan Le

Nov 12, 2018

Developer Documentation

1 Quickstart **1**
- 1.1 Installation . . . 1
- 1.2 Initializing Browseth . . . 1
- 1.3 Accounts . . . 1
- 1.4 Send requests . . . 2
- 1.5 Contract Instances . . . 2

2 Getting Started **3**
- 2.1 Installation . . . 3
- 2.2 Choosing an Ethereum RPC (Remote Procedure Call) . . . 3
- 2.3 Initializing Browseth . . . 3
- 2.4 Types of Requests . . . 4
- 2.5 Accounts . . . 4
- 2.6 Sending Ether . . . 4
- 2.7 Contract Instances . . . 4

3 Cookbook **5**
- 3.1 Contracts . . . 5

4 Browser **7**
- 4.1 Initialization . . . 7
- 4.2 Contract . . . 7
- 4.3 Units . . . 7
- 4.4 Accounts . . . 8
- 4.5 Utils . . . 8

5 Units **9**

6 Utilities **11**
- 6.1 Array Buffers . . . 11
- 6.2 Address . . . 12
- 6.3 Crypto . . . 12
- 6.4 Interval . . . 12
- 6.5 Param . . . 12
- 6.6 RLP . . . 12
- 6.7 Block Tracker . . . 13
- 6.8 Transaction Listener . . . 13

6.9 Observable . . . 14
6.10 Emitter . . . 14

7 Contracts **17**
7.1 Creating Instances . . . 17
7.2 Deploying Contracts . . . 17
7.3 Contract Functions . . . 18
7.4 Contract Events . . . 18

8 ENS **21**
8.1 Creating Instances . . . 21
8.2 Prototype . . . 21

9 Signers **23**
9.1 Private Key Signer . . . 23

CHAPTER 1

Quickstart

Browseth quickstart for those already familiar with Ethereum development. New to Ethereum? Check out the getting-started.

1.1 Installation

From your project directory:

```
yarn add @browseth/browser
```

Import inside relevant project files:

```
import Browseth from '@browseth/browser'
```

1.2 Initializing Browseth

Initialize Browseth with an Ethereum RPC url or web3 instance.

By default, Browseth uses http://localhost:8545.

```
const beth = new Browseth("https://mainnet.infura.io")
// or
const beth = new Browseth(window.web3)
```

1.3 Accounts

The following account types are supported: private key, ledger, and online.

```
import PrivateKeySigner from '@browseth/signer-private-key'
import SignerLedger from '@browseth/signer-ledger'

beth.useSignerAccount(new PrivateKeySigner(PRIVATE_KEY));
beth.useSignerAccount(new SignerLedger());
beth.useOnlineAccount();
```

1.4 Send requests

```
beth.send({ to: ADDRESS, value: beth.etherToWei('.01') });
```

1.5 Contract Instances

```
const contractInstance = beth.contract(contract.abi, {bin: contract.bin, address:␣
↪contract.address});

contractInstance.construct(params).send(); // deploying contract

contractInstance.fn
    .functionName(params)
    .call()
    .then(console.log); // function call

contractInstance.fn
    .functionName(params)
    .send({ value: beth.etherToWei('1') })
    .then(txHash => {
        beth.tx.listen(txHash).then(console.log)
    }); // send then log receipt
```

CHAPTER 2

Getting Started

Browseth is a simple JavaScript library for Ethereum.

2.1 Installation

From your project directory:

```
yarn add @browseth/browser
```

Import inside relevant project files:

```
import Browseth from '@browseth/browser'
```

2.2 Choosing an Ethereum RPC (Remote Procedure Call)

An Ethereum RPC is your gateway to interacting with Ethereum.

Ethereum nodes have the option to expose a JSON RPC allowing developers to interact with the Ethereum network.

A local Ethereum node usually exposes a JSON RPC at port 8545. There are services like Infura that provide a public JSON RPC for developers.

2.3 Initializing Browseth

Initialize Browseth with an Ethereum RPC url or web3 instance.

By default, Browseth uses http://localhost:8545.

```
const beth = new Browseth("https://mainnet.infura.io")
// or
const beth = new Browseth(window.web3)
```

Now Browseth is connected to the Ethereum network!

2.4 Types of Requests

There are two types of requests to Ethereum: read and writes.

A **call** request is free to call but may not add, remove, or change any data in the blockchain.

A **send** request requires a network fee, but may change the state of the blockchain. These methods must be made by a transaction and mined before any changes to the state are made. So these methods are subject to fluctuating gas prices, network congestion, and miner heuristics.

2.5 Accounts

Accounts are required to make send requests.

The following account types are supported: private key, ledger, and online.

```
import PrivateKeySigner from '@browseth/signer-private-key'
import SignerLedger from '@browseth/signer-ledger'

beth.useSignerAccount(new PrivateKeySigner(PRIVATE_KEY));
beth.useSignerAccount(new SignerLedger());
beth.useOnlineAccount();
```

2.6 Sending Ether

Once an account is connected to Browseth, you can make send requests.

```
beth.send({ to: ADDRESS, value: beth.etherToWei('.01') });
```

2.7 Contract Instances

To interact with contracts, we use contract instances made from our Browseth instance.

```
const contractInstance = beth.contract(contract.abi, {address: contract.address});

contractInstance.fn.functionName(params).call().then(console.log); // function call

contractInstance.fn.functionName(params)
    .send({ value: beth.etherToWei('1') })
    .then(txHash => {
        beth.tx.listen(txHash).then(console.log)
    }); // send then log receipt
```

CHAPTER 3

Cookbook

3.1 Contracts

Example Contract

```
pragma solidity ^0.4.22;

contract Test {
    uint256 a;

    event ASet(uint256 indexed a, uint256 aTimesTen);

    constructor(uint256 _a) public {
        a = _a;
    }

    function getA() public view returns (uint256) {
        return a;
    }

    function setA(uint256 _a) public payable {
        a = _a;
        emit ASet(a, a * 10);
    }
}
```

Compile the contract to get the contract ABI and binary

Creating Contract Instances

```
import testContractJson from './contract.json';

const testContractInstance = beth.contract(contract.abi, {bin: contract.bin});
```

Deploying Contracts

```
const a = 1231123;

testContractInstance
.construct(a)
.send()
.then(txHash => {
  beth.tx.listen(txHash)
    .then(receipt => console.log(receipt)
});
```

Contract calls

```
testContractInstance.fn.getA().call({to: contractAddress}).then(console.log)
```

Contract sends

```
testContractInstance.fn.setA(123123)
.send({to: contractAddress, value: beth.ethToWei('.01')})
.then(txHash => {
  beth.tx.listen(txHash)
    .then(receipt => console.log(receipt)
});
```

Read event logs

```
testContractInstance.ev.ASet()
.logs('earliest', 'latest')
.then(console.log)
```

Subscribe to Events

```
testContractInstance.ev.ASet()
.subscribe('latest')
.on(console.log)
```

CHAPTER 4

Browser

4.1 Initialization

```
import Browseth from '@browseth/browser';

const beth = new Browseth();
```

4.2 Contract

prototype **. contract (contractAbi [, options])** returns *Contracts* instance.

Options may have the properties:

- **bin** — contract binary (required for contract deployment)
- **address** — address of already deployed contract - .send() and .call() will default to this for the {to: address} option

4.3 Units

prototype **. convert (fromUnit, value, toUnit)** convert unit of value to unit

prototype **. etherToWei (value)** convert value in ether to wei

prototype **. gweiToWei (value)** convert value in gwei to wei

prototype **. weiToEther (value)** convert value in wei to ether

prototype **. toWei (fromUnit, value)** convert unit of value to wei

prototype **. toEther (fromUnit, value)** convert unit of value to ether

prototype **. unitToPow (unit)** returns the power of the unit relative to wei

4.4 Accounts

prototype . **useAccount (account)** switch to account

prototype . **useOnlineAccount (onlineAccount)** switch to online account

prototype . **useSignerAccount (signerAccount)** switch to signer account

prototype . **addOnlineAccount (onlineAccount)** adds online account to list of accounts

prototype . **addSignerAccount ( )** adds signer account to list of accounts

4.5 Utils

4.5.1 Addresses

prototype . **checksum (value)** Returns an address from bytes

prototype . **isValidAddress (value)** Checks if the given value is a valid address

4.5.2 Crypto

prototype . **keccak256 (value)** returns the keccak256 of a string

prototype . **namehash (name)** returns the node of a '.eth' domain string

4.5.3 Transaction Listener

prototype . **tx** *Transaction Listener* instance

4.5.4 Block Tracker

prototype . **block** *Block Tracker* instance

CHAPTER 5

Units

Unit conversion library

```
import * as units from '@browseth/units';
```

You can also import specific functions

```
import {etherToWei, conversion} from '@browseth/units';
```

---

units **. convert (fromUnit, value, toUnit)** convert unit of value to unit

units **. etherToWei (value)** convert value in ether to wei

units **. gweiToWei (value)** convert value in gwei to wei

units **. weiToEther (value)** convert value in wei to ether

units **. toWei (fromUnit, value)** convert unit of value to wei

units **. toEther (fromUnit, value)** convert unit of value to ether

units **. unitToPow (unit)** returns the power of the unit relative to wei

Supported Units: wei, kwei, ada, femtoether, mwei, babbage, picoether, gwei, shannon, nanoether, nano, szabo, microether, micro, finney, milliether, milli, ether, kether, grand, einstein, mether, gether, tether

CHAPTER 6

Utilities

```
const utils = require('@browseth/utils');
```

or

```
import utils from '@browseth/utils';
```

6.1 Array Buffers

An Array Buffer is an Array Buffer.

utils.ab **. isBytes (value [, length])** Checks to see if value is bytes and if it matches optional length

utils.ab **. fromView (view)** Returns an Array Buffer from view

utils.ab **. fromBytes (value [, length])** Returns Array Buffer from bytes with optional length

utils.ab **. fromUtf8 (value)** Returns Array Buffer from fromUtf8

utils.ab **. fromUInt (value)** Returns Array Buffer from UInt

utils.ab **. toUf8 (value)** Converts Array Buffer into Utf8

utils.ab **. toTwos (value, size)** Converts Array Buffer into a two's compliment

utils.ab **. stripStart (value)** Strips out the start of an Array Buffer

utils.ab **. padStart (value, length [, fillByte])** Pads the start of an Array Buffer

utils.ab **. padEnd (value, length [, fillByte])** Pads the end of an Array Buffer

utils.ab **. concat (values)** Concats an array of Array Buffers

6.2 Address

Utilities for manipulating addresses

utils.address **. isValid (value)** Checks if the given value is a valid address

utils.address **. from (value)** Returns an address from bytes

utils.address **. fromAddressAndNonce (address, nonce)** Returns an address from an address and nonce

6.3 Crypto

utils.crypto **. keccak256 (value)** returns the keccak256 of a string

utils.crypto **. namehash (name)** returns the node of a '.eth' domain string

6.4 Interval

utils.interval **. setUnrefedInterval (fn, delay [, args])** Sets an interval that dies when the function it's wrapped in is finished

utils.interval **. setUnrefedTimeout (fn, delay [, args])** Sets a timeout that dies when the function it's wrapped in is finished

6.5 Param

utils.param **. toData (value, length)** Converts parameters to hex

utils.param **. toQuantity (value)** Converts parameters to hex string quantity

utils.param **. toTag (value)** Converts value into a tag

utils.param **. isData (value [, length])** Checks if value is data of optional length

utils.param **. isQuantity (value)** Checks if value is a quantity

utils.param **. isTag (value)** Checks if value is a tag

utils.param **. fromData (value, length)** Converts value to uint8Array of length

utils.param **. fromQuantity (value)** Converts quantity to Big Number

utils.param **. fromTag (value)** Converts tag to Big Number

6.6 RLP

RLP (Recursive Length Prefix) is the main encoding method used to serialize objects in Ethereum

utils.rlp **. encode (value)** Encodes value to Array Buffer

utils.rlp **. encodeLength (len, offset)** Encodes length to Array Buffer with offset

6.7 Block Tracker

Poll for blocks every 5 seconds until a block number is confirmed. Use this class to keep track of block(s). Contains #emitter.

6.7.1 Creating Instances

new Browseth.utils **. BlockTracker (requestQueue [, confirmationDelay = 0])** Request queue is an eth reference. The confirmation delay is the minimum number of confirmed blocks until the block is considered confirmed.

6.7.2 Prototype

prototype **. addTracker (key [, options])** Track a block.

Options may have the following properties:

- **synced** – 'latest', 'earliest', or block # to track (defaults to 'latest')
- **confirmationDelay** – minimum # of confirmed blocks until tracked block is considered confirmed

prototype **. syncBlockNumber ( )** Sets the latest block number

emits 'block.number' with block # passed to the event callback

See #emitter

prototype **. syncBlocks ( )** Syncs blocks to latest block

emits 'block' for every synced block - block is passed to the event callback

See #emitter

6.8 Transaction Listener

Monitor transactions

6.8.1 Creating Instances

new Browseth.utils **. TxListener (ethRef)** Create new TxListener object with eth reference.

6.8.2 Prototype

prototype **. listen (txHash): <Promise>** Listen for a transaction until it is mined. Returns a promise that resolves to a transaction receipt.

If the listener does not see a receipt after 30 minutes it throws assuming the transaction has been dropped from the network

Listing 1: *Example*

```
import Browseth from '@browseth/browser'

const beth = new Browseth('https://mainnet.infura.io');
beth.useOnlineAccount();

const txListener = new Browseth.utils.TxListener(beth);

txListener.listen(txHash)
    .then(receipt => console.log(receipt))
    .catch(e => console.log('Transaction dropped!'))
```

6.9 Observable

Subscribe to value changes with callbacks

6.9.1 Creating Instances

new Browseth.utils **. Observable (value)** Create new Observable object with the value to watch.

6.9.2 Prototype

prototype **. subscribe (fn)** Add function to list of callbacks on value change. returns function to used unsubscribe function

prototype **. set (newValue)** Set the new value to watch. Triggers subscribed functions

prototype **. get ( )** Gets the current watched value.

Listing 2: *Example*

```
const observable = new Browseth.utils.Observable('123');

const unsubscribe = observable.subscribe(() => console.log('This is an example'));

observable.set('456');  // Sets new value and logs 'This is an example'

unsubscribe(); // unsubscribe earlier subscribed function

observable.set('78'); // Will set new value with no callbacks

observable.get(); // returns '78'
```

6.10 Emitter

Add events with callbacks and trigger those callbacks by emitting events.

6.10.1 Creating Instances

new Browseth.utils **. Emitter ( )** Create new Emitter object.

6.10.2 Prototype

prototype **. on (event, fn)** Add event label and provide callback

prototype **. off (event, fn)** Remove callback from an event

prototype **. onEvery (fn)** Provide callback for every emit

prototype **. emit (event [, params])** Emit an event and pass parameters to the callbacks

Listing 3: *Example*

```
const emitter = new Browseth.utils.Emitter('123');

emitter.on('test', () => console.log('example'));

emitter.onEvery(() => console.log('example2'));

emitter.emit('test') // Console logs 'example' and 'example2'
```

CHAPTER 7

Contracts

There are two types of methods that can be called on a Contract:

A **call** method may not add, remove or change any data in the storage. These methods are free to call.

A **send** method requires a fee, but may change the state of the blockchain or any data in the storage. These methods must be made by a transaction and mined before any changes to the state are made. Therefore, these methods are subject to fluctuating gas prices, network congestion, and miner heuristics.

```
import Contract from '@browseth/contract';
```

7.1 Creating Instances

new Contract **(ethRef, contractAbi [, options])** Options may have the properties:

- **bin** — contract binary (required for contract deployment)
- **address** — address of already deployed contract - .send() and .call() will default to this for the {to: address} option

7.2 Deploying Contracts

prototype **. construct ([params])** Takes in constructor parameters for the deploying contract returns send() and gas() methods

. send ([options]) deploys contract and returns promise resolving transaction hash

. gas ([options]) returns the estimated gas for deploying the contract

Options may have the properties:

- **chainId** — set contract binary for contract deployment
- **gasPrice** — set gas price in wei for transaction

- **gas** — sets the max amount of gas for the transaction

```
import Browseth from '@browseth/browser'
import Contract from '@browseth/contract'
import PrivateKeySigner from '@browseth/signer-private-key'

const beth = new Browseth(eth_rpc);
beth.useSignerAccount(new PrivateKeySigner(PRIVATE_KEY));

const contractInstance = new Contract(beth, contract.abi, {bin: contract.bin});
const txHash = await contractInstance.construct().send({ gasPrice: 10000000000});
```

7.3 Contract Functions

prototype **. fn . functionName ([params])** Takes in parameters for calling contract function.

returns send() and call() methods.

. send ([options]) makes write call to contract function.

returns promise resolving transaction hash.

. call ([options]) makes readonly call to contract function

Options may have the properties:

- **chainId** — set contract binary for contract deployment
- **gasPrice** — set gas price in wei for transaction
- **gas** — sets the max amount of gas for the transaction
- **to** — sets the address of where to send call to (defaults to 'address' in initialization)

```
const txHash = testContractInstance.fn
    .setA(0x123123123)
    .send();
```

7.4 Contract Events

prototype **. ev . eventName ([indexed params])** Optional indexed parameter values that event log must match.

returns logs() and subscribe() methods.

. logs (fromBlock, toBlock, contractAddress) Get logs of contract with block range.

- **fromBlock** — blockNumber (defaults to 'earliest')
- **toBlock** — blockNumber (defaults to 'latest')

. subscribe (fromBlock, contractAddress) Subscribe to contract events with callback.

- **fromBlock** — blockNumber (defaults to 'latest')

returns .on() method

.on (callback) Calls callback function when event occurs.

```
contractInstance.ev
    .ASet({ a: 0x123123123 })
    .logs('earliest', 'latest', contractAddress)
    .then(console.log)

contractInstance.ev
    .ASet()
    .subscribe('earliest', contractAddress)
    .on(console.log)
```

CHAPTER 8

ENS

Ethereum Name Service (ENS) library for name lookup and standard resolver interface reading.

```
import EnsLookup from '@browseth/units';
```

8.1 Creating Instances

new EnsLookup **(ethRef)** Initialize EnsLookup object with browseth instance

8.2 Prototype

prototype **. resolverAddress (node)** Returns the address of the node's resolver

prototype **. address (node)** Returns the address field set in the node's resolver

prototype **. name (node)** Returns the name set in the node's resolver

prototype **. text (node, key)** Returns the text of a key in the node's resolver

prototype **. allTexts (node, key)** Returns all the text changes of a key in the node's resolver

prototype **. supportsInterface (node, interfaceId)** Checks if the interfaceId is supported by the node's resolver

CHAPTER 9

Signers

A **Signer** manages a private/public key pair which is used to cryptographically sign transactions and prove ownership on the Ethereum network.

9.1 Private Key Signer

```
const PrivateKeySigner = require('@browseth/signer-private-key')
```

or

```
import PrivateKeySigner from '@browseth/signer-private-key'
```

9.1.1 Creating Instances

new PrivateKeySigner **(privateKey)** Creates a private key signer object from *privateKey*

9.1.2 Prototype

prototype **. address ( )** Returns the address of the signer generated from the privateKey

prototype **. signMessage (message)** Returns a signed message

prototype **. signTransaction ([params])** Returns a signed transaction

Parameters may include:

- **to**
- **gasPrice**
- **gasLimit**

- **nonce**
- **data**
- **value**
- **chainId**