web3.js Documentation

Release 1.0.0

Fabian Vogelsteller, Marek Kotewicz, Jeffrey Wilcke, Marian Oance

Aug 06, 2019

User Documentation

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

web3.js is a collection of libraries which allow you to interact with a local or remote Ethereum node, using an HTTP, WebSocket or IPC connection.

The following documentation will guide you through *installing and running web3.js*, as well as providing a API reference documentation with examples.

Contents:

Keyword Index, Search Page

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 1

Getting Started

The web3.js library is a collection of modules which contain specific functionality for the Ethereum ecosystem.

- The `web3-eth` is for the Ethereum blockchain and smart contracts
- The `web3-shh` is for the whisper protocol to communicate p2p and broadcast
- The `web3-utils` contains useful helper functions for DApp developers.

1.1 Adding web3.js

First you need to get web3.js into your project. This can be done using the following methods:

- npm: `npm install web3`

After that you need to create a web3 instance and set a provider. A Ethereum compatible browser will have a `window.ethereum` or `web3.currentProvider` available. For web3.js, check `Web3.givenProvider`. If this property is `null` you should connect to your own local or remote node.

```
// in node.js use: const Web3 = require('web3');

// use the given Provider, e.g in the browser with Metamask, or instantiate a new␣
↪websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546', null, {});

// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'), null, {});

// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net, {}); // mac os␣
↪path
// or
```

(continues on next page)

(continued from previous page)

```
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net, {})); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

That's it! now you can use the `web3` object.

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 2

Callbacks Promises Events

To help web3 integrate into all kind of projects with different standards we provide multiple ways to act on asynchronous functions.

Most web3.js objects allow a callback as the last parameter, as well as returning promises to chain functions.

Ethereum as a blockchain has different levels of finality and therefore needs to return multiple "stages" of an action. To cope with requirement we return a "PromiEvent" for functions like *web3.eth.sendTransaction* or contract methods. These stages are encapsulated into a "PromiEvent", which combines a promise with an event emitter. The event emitter fires an event for each of the finality stages.

An example of a function that benefits from a PromiEvent is the *web3.eth.sendTransaction* method.

```
web3.eth.sendTransaction({from: '0x123...', data: '0x432...'})
.once('transactionHash', function(hash){ ... })
.once('receipt', function(receipt){ ... })
.on('confirmation', function(confNumber, receipt){ ... })
.on('error', function(error){ ... })
.then(function(receipt){
    // will be fired once the receipt is mined
});
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 3

Glossary

3.1 Specification

Functions:

- `type`: `"function"`, `"constructor"` (can be omitted, defaulting to `"function"`; `"fallback"` also possible but not relevant in web3.js);
- `name`: the name of the function (only present for function types);
- `constant`: `true` if function is specified to not modify the blockchain state;
- `payable`: `true` if function accepts ether, defaults to `false`;
- `stateMutability`: a string with one of the following values: `pure` (specified to not read blockchain state), `view` (same as `constant` above), `nonpayable` and `payable` (same as `payable` above);
- `inputs`: an array of objects, each of which contains:
  - `name`: the name of the parameter;
  - `type`: the canonical type of the parameter.
- `outputs`: an array of objects same as `inputs`, can be omitted if no outputs exist.

Events:

- `type`: always `"event"`
- `name`: the name of the event;
- `inputs`: an array of objects, each of which contains:
  - `name`: the name of the parameter;
  - `type`: the canonical type of the parameter.
  - `indexed`: `true` if the field is part of the log's topics, `false` if it one of the log's data segment.
- `anonymous`: `true` if the event was declared as `anonymous`.

3.2 Example

```
contract Test {
    uint a;
    address d = 0x1234567890123456789012345678901234567890;

    function Test(uint testInt)  { a = testInt;}

    event Event(uint indexed b, bytes32 c);

    event Event2(uint indexed b, bytes32 c);

    function foo(uint b, bytes32 c) returns(address) {
        Event(b, c);
        return d;
    }
}

// would result in the JSON:
[{
    "type":"constructor",
    "payable":false,
    "stateMutability":"nonpayable"
    "inputs":[{"name":"testInt","type":"uint256"}],
  },{
    "type":"function",
    "name":"foo",
    "constant":false,
    "payable":false,
    "stateMutability":"nonpayable",
    "inputs":[{"name":"b","type":"uint256"}, {"name":"c","type":"bytes32"}],
    "outputs":[{"name":"","type":"address"}]
  },{
    "type":"event",
    "name":"Event",
    "inputs":[{"indexed":true,"name":"b","type":"uint256"}, {"indexed":false,"name":"c
↪","type":"bytes32"}],
    "anonymous":false
  },{
    "type":"event",
    "name":"Event2",
    "inputs":[{"indexed":true,"name":"b","type":"uint256"},{"indexed":false,"name":"c
↪","type":"bytes32"}],
    "anonymous":false
}]
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 4

Web3

The Web3 class is a wrapper to house all Ethereum related modules.

4.1 Initiating of Web3

4.1.1 Parameters

1. `provider` - `string|object`: A URL or one of the Web3 provider classes.
2. `net` - `net.Socket` (optional): The net NodeJS package.
3. `options` - `object` (optional) The Web3 *options*

4.1.2 Example

```
import Web3 from 'web3';

// "Web3.givenProvider" will be set in a Ethereum supported browser.
const web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪net, options);

> web3.eth
> web3.shh
> web3.utils
> web3.version
```

4.2 Web3.modules

This Static property will return an object with the classes of all major sub modules, to be able to instantiate them manually.

4.2.1 Returns

`Object`: A list of modules:

- `Eth - Function`: the Eth module for interacting with the Ethereum network see *web3.eth* for more.
- `Net - Function`: the Net module for interacting with network properties see *web3.eth.net* for more.
- `Personal - Function`: the Personal module for interacting with the Ethereum accounts see *web3.eth.personal* for more.
- `Shh - Function`: the Shh module for interacting with the whisper protocol see *web3.shh* for more.

4.2.2 Example

```
Web3.modules
> {
    Eth(provider, net?, options?),
    Net(provider, net?, options?),
    Personal(provider, net?, options?),
    Shh(provider, net?, options?),
}
```

4.3 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

4.3.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

4.3.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

4.4 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

4.4.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block
- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

4.5 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

4.5.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

4.6 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

4.6.1 Returns

`string|number`: The current value of the defaultGasPrice property.

4.7 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

4.7.1 Returns

`string|number`: The current value of the defaultGas property.

4.8 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

4.8.1 Returns

`number`: The current value of transactionBlockTimeout

4.9 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

4.9.1 Returns

`number`: The current value of transactionConfirmationBlocks

4.10 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

4.10.1 Returns

`number`: The current value of transactionPollingTimeout

4.11 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

4.11.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

4.12 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

4.12.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

4.12.2 Returns

`Boolean`

4.12.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

4.13 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

4.13.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

4.13.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));
```

(continues on next page)

(continued from previous page)

```
// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

4.14 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

4.14.1 Returns

`Object`: The given provider set or `false`.

4.14.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

4.15 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

4.15.1 Returns

`Object`: The current provider set.

4.15.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

4.16 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

4.16.1 Parameters

none

4.16.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

4.16.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

4.17 version

Property of the Web3 class.

```
web3.version
```

Contains the version of the `web3` wrapper class.

4.17.1 Returns

`String`: The current version.

4.17.2 Example

```
web3.version;
> "1.0.0"
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 5

web3.eth

The `web3-eth` package allows you to interact with an Ethereum blockchain itself and the deployed smart contracts.

```
import Web3 from 'web3';
import {Eth} from 'web3-eth';

// "Web3.givenProvider" will be set if in an Ethereum supported browser.
const eth = new Eth(Web3.givenProvider || 'ws://some.local-or-remote.node:8546', null,
↪ options);


// or using the web3 umbrella package

const web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);

// -> web3.eth
```

5.1 Note on checksum addresses

All Ethereum addresses returned by functions of this package are returned as checksum addresses. This means some letters are uppercase and some are lowercase. Based on that it will calculate a checksum for the address and prove its correctness. Incorrect checksum addresses will throw an error when passed into functions. If you want to circumvent the checksum check you can make an address all lower- or uppercase.

5.1.1 Example

```
web3.eth.getAccounts(console.log);
> ["0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe" ,
↪"0x85F43D8a49eeB85d32Cf465507DD71d507100C1d"]
```

5.2 subscribe

For `web3.eth.subscribe` see the *Subscribe reference documentation*

5.3 Contract

For `web3.eth.Contract` see the *Contract reference documentation*

5.4 Iban

For `web3.eth.Iban` see the *Iban reference documentation*

5.5 personal

For `web3.eth.personal` see the *personal reference documentation*

5.6 accounts

For `web3.eth.accounts` see the *accounts reference documentation*

5.7 ens

For `web3.eth.ens` see the *Ens reference documentation*

5.8 abi

For `web3.eth.abi` see the *ABI reference documentation*

5.9 net

For `web3.eth.net` see the *net reference documentation*

5.10 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

5.10.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

5.10.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

5.11 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

5.11.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block
- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

5.12 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

5.12.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

5.13 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

5.13.1 Returns

`string|number`: The current value of the defaultGasPrice property.

5.14 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

5.14.1 Returns

`string|number`: The current value of the defaultGas property.

5.15 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

5.15.1 Returns

`number`: The current value of transactionBlockTimeout

5.16 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

5.16.1 Returns

`number`: The current value of transactionConfirmationBlocks

5.17 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

5.17.1 Returns

`number`: The current value of transactionPollingTimeout

5.18 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

5.18.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

5.19 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

5.19.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

5.19.2 Returns

`Boolean`

5.19.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

5.20 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

5.20.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

5.20.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

5.21 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

5.21.1 Returns

`Object`: The given provider set or `false`.

5.21.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

5.22 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

5.22.1 Returns

`Object`: The current provider set.

5.22.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

5.23 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

5.23.1 Parameters

none

5.23.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

5.23.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

5.24 getProtocolVersion

```
web3.eth.getProtocolVersion([callback])
```

Returns the Ethereum protocol version of the node.

5.24.1 Returns

`Promise<string>` - The protocol version.

5.24.2 Example

```
web3.eth.getProtocolVersion().then(console.log);
> "63"
```

5.25 isSyncing

```
web3.eth.isSyncing([callback])
```

Checks if the node is currently syncing and returns either a syncing object, or `false`.

5.25.1 Returns

`Promise<object|boolean>` - A sync object when the node is currently syncing or `false`:

- `startingBlock` - `Number`: The block number where the sync started.
- `currentBlock` - `Number`: The block number where at which block the node currently synced to already.
- `highestBlock` - `Number`: The estimated block number to sync to.
- `knownStates` - `Number`: The estimated states to download
- `pulledStates` - `Number`: The already downloaded states

5.25.2 Example

```
web3.eth.isSyncing()
.then(console.log);

> {
    startingBlock: 100,
    currentBlock: 312,
    highestBlock: 512,
    knownStates: 234566,
    pulledStates: 123455
}
```

5.26 getCoinbase

```
web3.eth.getCoinbase([callback])
```

Returns the coinbase address to which mining rewards will go.

5.26.1 Returns

`Promise<string>` - The coinbase address set in the node for mining rewards.

5.26.2 Example

```
web3.eth.getCoinbase().then(console.log);
> "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe"
```

5.27 isMining

```
web3.eth.isMining([callback])
```

Checks whether the node is mining or not.

5.27.1 Returns

`Promise<boolean>` - Returns `true` if the node is mining, otherwise `false`.

5.27.2 Example

```
web3.eth.isMining().then(console.log);
> true
```

5.28 getHashrate

```
web3.eth.getHashrate([callback])
```

Returns the number of hashes per second that the node is mining with.

5.28.1 Returns

`Promise<number>` - The Number of hashes per second.

5.28.2 Example

```
web3.eth.getHashrate().then(console.log);
> 493736
```

5.29 getGasPrice

```
web3.eth.getGasPrice([callback])
```

Returns the current gas price oracle. The gas price is determined by the last few blocks median gas price. GasPrice is the wei per unit of gas,.

5.29.1 Returns

`Promise<string>` - Number string of the current gas price in wei.

See the *A note on dealing with big numbers in JavaScript*.

5.29.2 Example

```
web3.eth.getGasPrice().then(console.log);
> "20000000000"
```

5.30 getAccounts

```
web3.eth.getAccounts([callback])
```

Will return a list of the unlocked accounts in the Web3 wallet or it will return the accounts from the currently connected node.

This means you can add accounts with *web3.eth.accounts.create()* and you will get them returned here.

5.30.1 Returns

`Promise<Array>` - An array of addresses controlled by node.

5.30.2 Example

```
web3.eth.getAccounts().then(console.log);
> ["0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
↪"0xDCc6960376d6C6dEa93647383FfB245CfCed97Cf"]
```

5.31 getBlockNumber

```
web3.eth.getBlockNumber([callback])
```

Returns the current block number.

5.31.1 Returns

`Promise<number>` - The number of the most recent block.

5.31.2 Example

```
web3.eth.getBlockNumber().then(console.log);
> 2744
```

5.32 getBalance

```
web3.eth.getBalance(address [, defaultBlock] [, callback])
```

Get the balance of an address at a given block.

5.32.1 Parameters

1. `String` - The address to get the balance of.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with web3.eth.defaultBlock.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.32.2 Returns

`Promise<string>` - The current balance for the given address in wei.

See the A note on dealing with big numbers in JavaScript.

5.32.3 Example

```
web3.eth.getBalance("0x407d73d8a49eeb85d32cf465507dd71d507100c1").then(console.log);
> "1000000000000"
```

5.33 getStorageAt

```
web3.eth.getStorageAt(address, position [, defaultBlock] [, callback])
```

Get the storage at a specific position of an address.

5.33.1 Parameters

1. `String` - The address to get the storage from.
2. `Number` - The index position of the storage.
3. `Number|String` - (optional) If you pass this parameter it will not use the default block set with web3.eth.defaultBlock.
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.33.2 Returns

`Promise<string>` - The value in storage at the given position.

5.33.3 Example

```
web3.eth.getStorageAt("0x407d73d8a49eeb85d32cf465507dd71d507100c1", 0).then(console.
↪log);
> "0x033456732123ffff2342342dd12342434324234234fd234fd23fd4f23d4234"
```

5.34 getCode

```
web3.eth.getCode(address [, defaultBlock] [, callback])
```

Get the code at a specific address.

5.34.1 Parameters

1. `String` - The address to get the code from.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with web3.eth.defaultBlock.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.34.2 Returns

`Promise<string>` - The data at given address `address`.

5.34.3 Example

```
web3.eth.getCode("0xd5677cf67b5aa051bb40496e68ad359eb97cfbf8").then(console.log);
>
↪"0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060078202905091900
↪"
```

5.35 getBlock

```
web3.eth.getBlock(blockHashOrBlockNumber [, returnTransactionObjects] [, callback])
```

Returns a block matching the block number or block hash.

5.35.1 Parameters

1. `String|Number` - The block number or block hash. Or the string `"genesis"`, `"latest"` or `"pending"` as in the default block parameter.
2. `Boolean` - (optional, default `false`) If `true`, the returned block will contain all transactions as objects, if `false` it will only contains the transaction hashes.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.35.2 Returns

`Promise<object>` - The block object:

- `number` - `Number`: The block number. `null` when its pending block.
- `hash` 32 Bytes - `String`: Hash of the block. `null` when its pending block.
- `parentHash` 32 Bytes - `String`: Hash of the parent block.
- `nonce` 8 Bytes - `String`: Hash of the generated proof-of-work. `null` when its pending block.
- `sha3Uncles` 32 Bytes - `String`: SHA3 of the uncles data in the block.
- `logsBloom` 256 Bytes - `String`: The bloom filter for the logs of the block. `null` when its pending block.
- `transactionsRoot` 32 Bytes - `String`: The root of the transaction trie of the block
- `stateRoot` 32 Bytes - `String`: The root of the final state trie of the block.
- `receiptsRoot` 32 Bytes - `String`: Transaction receipts are used to store the state after a transaction has been executed and are kept in an index-keyed trie. The hash of its root is placed in the block header as the receipts root.
- `miner` - `String`: The address of the beneficiary to whom the mining rewards were given.
- `difficulty` - `String`: Integer of the difficulty for this block.

- `totalDifficulty` - `String`: Integer of the total difficulty of the chain until this block.
- `extraData` - `String`: The "extra data" field of this block.
- `size` - `Number`: Integer the size of this block in bytes.
- `gasLimit` - `Number`: The maximum gas allowed in this block.
- `gasUsed` - `Number`: The total used gas by all transactions in this block.
- `timestamp` - `Number | String`: The unix timestamp for when the block was collated (returns a string if a overflow got detected).
- `transactions` - `Array`: Array of transaction objects, or 32 Bytes transaction hashes depending on the `returnTransactionObjects` parameter.
- `uncles` - `Array`: Array of uncle hashes.

5.35.3 Example

```
web3.eth.getBlock(3150).then(console.log);
> {
    "number": 3,
    "hash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
    "parentHash": "0x2302e1c0b972d00932deb5dab9eb2982f570597d9d42504c05d9c2147eaf9c88
↪",
    "nonce": "0xfb6e1a62d119228b",
    "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
↪",
    "logsBloom":
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "transactionsRoot":
↪"0x3a1b03875115b79539e5bd33fb00d8f7b7cd61929d5a3c574f507b8acf415bee",
    "stateRoot": "0xf1133199d44695dfa8fd1bcfe424d82854b5cebef75bddd7e40ea94cda515bcb",
    "receiptsRoot: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421
↪',
    "miner": "0x8888f1f195afa192cfee860698584c030f4c9db1",
    "difficulty": '21345678965432',
    "totalDifficulty": '324567845321',
    "size": 616,
    "extraData": "0x",
    "gasLimit": 3141592,
    "gasUsed": 21662,
    "timestamp": 1429287689,
    "transactions": [
        "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b"
    ],
    "uncles": []
}
```

5.36 getBlockTransactionCount

```
web3.eth.getBlockTransactionCount(blockHashOrBlockNumber [, callback])
```

Returns the number of transaction in a given block.

5.36.1 Parameters

1. `String|Number` - The block number or hash. Or the string `"genesis"`, `"latest"` or `"pending"` as in the default block parameter.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.36.2 Returns

`Promise<number>` - The number of transactions in the given block.

5.36.3 Example

```
web3.eth.getBlockTransactionCount("0x407d73d8a49eeb85d32cf465507dd71d507100c1").
↪then(console.log);
> 1
```

5.37 getUncle

```
web3.eth.getUncle(blockHashOrBlockNumber, uncleIndex [, callback])
```

Returns a blocks uncle by a given uncle index position.

5.37.1 Parameters

1. `String|Number` - The block number or hash. Or the string `"genesis"`, `"latest"` or `"pending"` as in the default block parameter.
2. `Number` - The index position of the uncle.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.37.2 Returns

`Promise<object>` - The returned uncle. For a return value see *web3.eth.getBlock()*.

Note: An uncle doesn't contain individual transactions.

5.37.3 Example

```
web3.eth.getUncle(500, 0).then(console.log);
> // see web3.eth.getBlock
```

5.38 getTransaction

```
web3.eth.getTransaction(transactionHash [, callback])
```

Returns a transaction matching the given transaction hash.

5.38.1 Parameters

1. `String` - The transaction hash.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.38.2 Returns

`Promise<object>` - A transaction object its hash `transactionHash`:

- `hash` 32 Bytes - `String`: Hash of the transaction.
- `nonce` - `Number`: The number of transactions made by the sender prior to this one.
- `blockHash` 32 Bytes - `String`: Hash of the block where this transaction was in. `null` when its pending.
- `blockNumber` - `Number`: Block number where this transaction was in. `null` when its pending.
- `transactionIndex` - `Number`: Integer of the transactions index position in the block. `null` when its pending.
- `from` - `String`: Address of the sender.
- `to` - `String`: Address of the receiver. `null` when its a contract creation transaction.
- `value` - `String`: Value transferred in wei.
- `gasPrice` - `String`: The wei per unit of gas provided by the sender in wei.
- `gas` - `Number`: Gas provided by the sender.
- `input` - `String`: The data sent along with the transaction.

5.38.3 Example

```
web3.eth.getTransaction(
↪'0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b§234').
↪then(console.log);
> {
    "hash": "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
    "nonce": 2,
    "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
    "blockNumber": 3,
    "transactionIndex": 0,
    "from": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",
    "to": "0x6295ee1b4f6dd65047762f924ecd367c17eabf8f",
    "value": '123450000000000000',
    "gas": 314159,
    "gasPrice": '2000000000000',
    "input": "0x57cb2fc4"
}
```

5.39 getPendingTransactions

```
web3.eth.getPendingTransactions([, callback])
```

Returns a list of pending transactions.

5.39.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.39.2 Returns

`Promise<object[]>` - Array of pending transactions:

- `hash` 32 Bytes - `String`: Hash of the transaction.
- `nonce` - `Number`: The number of transactions made by the sender prior to this one.
- `blockHash` 32 Bytes - `String`: Hash of the block where this transaction was in. `null` when its pending.
- `blockNumber` - `Number`: Block number where this transaction was in. `null` when its pending.
- `transactionIndex` - `Number`: Integer of the transactions index position in the block. `null` when its pending.
- `from` - `String`: Address of the sender.
- `to` - `String`: Address of the receiver. `null` when its a contract creation transaction.
- `value` - `String`: Value transferred in wei.
- `gasPrice` - `String`: The wei per unit of gas provided by the sender in wei.
- `gas` - `Number`: Gas provided by the sender.
- `input` - `String`: The data sent along with the transaction.

5.39.3 Example

```
web3.eth.getPendingTransactions().then(console.log);
> [
    {
        hash: '0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b',
        nonce: 2,
        blockHash:
↪'0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46',
        blockNumber: 3,
        transactionIndex: 0,
        from: '0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b',
        to: '0x6295ee1b4f6dd65047762f924ecd367c17eabf8f',
        value: '123450000000000000',
        gas: 314159,
        gasPrice: '2000000000000',
```

(continues on next page)

(continued from previous page)

```
        input: '0x57cb2fc4'
        v: '0x3d',
        r: '0xaabc9ddafffb2ae0bac4107697547d22d9383667d9e97f5409dd6881ce08f13f',
        s: '0x69e43116be8f842dcd4a0b2f760043737a59534430b762317db21d9ac8c5034'
    },....,{
        hash: '0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b',
        nonce: 3,
        blockHash:
↪'0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46',
        blockNumber: 4,
        transactionIndex: 0,
        from: '0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b',
        to: '0x6295ee1b4f6dd65047762f924ecd367c17eabf8f',
        value: '123450000000000000',
        gas: 314159,
        gasPrice: '2000000000000',
        input: '0x57cb2fc4'
        v: '0x3d',
        r: '0xaabc9ddafffb2ae0bac4107697547d22d9383667d9e97f5409dd6881ce08f13f',
        s: '0x69e43116be8f842dcd4a0b2f760043737a59534430b762317db21d9ac8c5034'
    }
]
```

5.40 getTransactionFromBlock

```
getTransactionFromBlock(hashStringOrNumber, indexNumber [, callback])
```

Returns a transaction based on a block hash or number and the transactions index position.

5.40.1 Parameters

1. `String` - A block number or hash. Or the string `"genesis"`, `"latest"` or `"pending"` as in the default block parameter.
2. `Number` - The transactions index position.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.40.2 Returns

`Promise<object>` - A transaction object, see *web3.eth.getTransaction*:

5.40.3 Example

```
const transaction = web3.eth.getTransactionFromBlock('0x4534534534', 2).then(console.
↪log);
> // see web3.eth.getTransaction
```

5.41 getTransactionReceipt

```
web3.eth.getTransactionReceipt(hash [, callback])
```

Returns the receipt of a transaction by transaction hash.

Note: The receipt is not available for pending transactions and returns `null`.

5.41.1 Parameters

1. `String` - The transaction hash.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.41.2 Returns

`Promise` returns `Object` - A transaction receipt object, or `null` when no receipt was found:

- `status` - `Boolean`: `TRUE` if the transaction was successful, `FALSE`, if the EVM reverted the transaction.
- `blockHash` 32 Bytes - `String`: Hash of the block where this transaction was in.
- `blockNumber` - `Number`: Block number where this transaction was in.
- `transactionHash` 32 Bytes - `String`: Hash of the transaction.
- `transactionIndex`- `Number`: Integer of the transactions index position in the block.
- `from` - `String`: Address of the sender.
- `to` - `String`: Address of the receiver. `null` when its a contract creation transaction.
- `contractAddress` - `String`: The contract address created, if the transaction was a contract creation, otherwise `null`.
- `cumulativeGasUsed` - `Number`: The total amount of gas used when this transaction was executed in the block.
- `gasUsed`- `Number`: The amount of gas used by this specific transaction alone.
- `logs` - `Array`: Array of log objects, which this transaction generated.

5.41.3 Example

```
const receipt = web3.eth.getTransactionReceipt(
↪'0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b')
                        .then(console.log);
> {
  "status": true,
  "transactionHash":
↪"0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
  "transactionIndex": 0,
  "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "blockNumber": 3,
  "contractAddress": "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
```

(continues on next page)

(continued from previous page)

```
  "cumulativeGasUsed": 314159,
  "gasUsed": 30234,
  "logs": [{
         // logs as returned by getPastLogs, etc.
     }, ...]
}
```

5.42 getTransactionCount

```
web3.eth.getTransactionCount(address [, defaultBlock] [, callback])
```

Get the numbers of transactions sent from this address.

5.42.1 Parameters

1. `String` - The address to get the numbers of transactions from.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with web3.eth.defaultBlock.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.42.2 Returns

`Promise<number>` - The number of transactions sent from the given address.

5.42.3 Example

```
web3.eth.getTransactionCount("0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe").
↪then(console.log);
> 1
```

5.43 sendTransaction

```
web3.eth.sendTransaction(transactionObject [, callback])
```

Sends a transaction to the network.

5.43.1 Parameters

1. `Object` - The transaction object to send:

- `from` - `String|Number`: The address for the sending account. Uses the web3.eth.defaultAccount property, if not specified. Or an address or index of a local wallet in web3.eth.accounts.wallet.
- `to` - `String`: (optional) The destination address of the message, left undefined for a contract-creation transaction.
- `value` - `Number|String|BN|BigNumber`: (optional) The value transferred for the transaction in wei, also the endowment if it's a contract-creation transaction.
- `gas` - `Number`: (optional, default: To-Be-Determined) The amount of gas to use for the transaction (unused gas is refunded).
- `gasPrice` - `Number|String|BN|BigNumber`: (optional) The price of gas for this transaction in wei, defaults to *web3.eth.gasPrice*.
- `data` - `String`: (optional) Either a ABI byte string containing the data of the function call on a contract, or in the case of a contract-creation transaction the initialisation code.
- `nonce` - `Number`: (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

2. `callback` - `Function`: (optional) Optional callback, returns an error object as first parameter and the result as second.

Note: The `from` property can also be an address or index from the web3.eth.accounts.wallet. It will then sign locally using the private key of that account, and send the transaction via *web3.eth.sendSignedTransaction()*.

5.43.2 Returns

The **callback** will return the 32 bytes transaction hash.

`PromiEvent`: A *promise combined event emitter*. Will be resolved when the transaction *receipt* is available. Additionally the following events are available:

- `"transactionHash"` returns `String`: Is fired right after the transaction is sent and a transaction hash is available.
- `"receipt"` returns `Object`: Is fired when the transaction receipt is available.
- `"confirmation"` returns `Number`, `Object`: Is fired for every confirmation up to the 12th confirmation. Receives the confirmation number as the first and the *receipt* as the second argument. Fired from confirmation 0 on, which is the block where its minded.
- `"error"` returns `Error`: Is fired if an error occurs during sending. If a out of gas error, the second parameter is the receipt.

5.43.3 Example

```
// compiled solidity source code using https://remix.ethereum.org
const code =
↪"603d80600c6000396000f3007c0100000000000000000000000000000000000000000000000000000000600035046
↪";
```

(continues on next page)

(continued from previous page)

```
// using the callback
web3.eth.sendTransaction({
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    data: code // deploying a contract
}, function(error, hash){
    ...
});

// using the promise
web3.eth.sendTransaction({
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    to: '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe',
    value: '1000000000000000'
})
.then(function(receipt){
    ...
});


// using the event emitter
web3.eth.sendTransaction({
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    to: '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe',
    value: '1000000000000000'
})
.on('transactionHash', function(hash){
    ...
})
.on('receipt', function(receipt){
    ...
})
.on('confirmation', function(confirmationNumber, receipt){ ... })
.on('error', console.error); // If a out of gas error, the second parameter is the␣
↪receipt.
```

5.44 sendSignedTransaction

```
web3.eth.sendSignedTransaction(signedTransactionData [, callback])
```

Sends an already signed transaction, generated for example using *web3.eth.accounts.signTransaction*.

5.44.1 Parameters

1. `String` - Signed transaction data in HEX format
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.44.2 Returns

`PromiEvent`: A *promise combined event emitter*. Will be resolved when the transaction *receipt* is available.

Please see the return values for *web3.eth.sendTransaction* for details.

5.44.3 Example

```
const Tx = require('ethereumjs-tx');
const privateKey = new Buffer(
↪'e331b6d69882b4cb4ea581d88e0b604039a3de5967688d3dcffdd2270c0fd109', 'hex')

const rawTx = {
  nonce: '0x00',
  gasPrice: '0x09184e72a000',
  gasLimit: '0x2710',
  to: '0x0000000000000000000000000000000000000000',
  value: '0x00',
  data: '0x7f7465737432000000000000000000000000000000000000000000000000000000000000600057'
}

const tx = new Tx(rawTx);
tx.sign(privateKey);

const serializedTx = tx.serialize();

// console.log(serializedTx.toString('hex'));
//␣
↪0xf889808609184e72a00082271094000000000000000000000000000000000000000080a47f74657374320000000000000
web3.eth.sendSignedTransaction('0x' + serializedTx.toString('hex'))
.on('receipt', console.log);

> // see eth.getTransactionReceipt() for details
```

5.45 sign

```
web3.eth.sign(dataToSign, address [, callback])
```

Signs data using a specific account. This account needs to be unlocked.

5.45.1 Parameters

1. `String` - Data to sign. If String it will be converted using *web3.utils.utf8ToHex*.
2. `String|Number` - Address to sign data with. Or an address or index of a local wallet in web3.eth.accounts.wallet.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

Note: The 2. `address` parameter can also be an address or index from the web3.eth.accounts.wallet. It will then sign locally using the private key of this account.

5.45.2 Returns

`Promise<string>` - The signature.

5.45.3 Example

```
web3.eth.sign("Hello world", "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪"

// the below is the same
web3.eth.sign(web3.utils.utf8ToHex("Hello world"),
↪"0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪"
```

5.46 signTransaction

```
web3.eth.signTransaction(transactionObject [, address,] [, callback])
```

Signs a transaction with the private key of the given address. If the given address is a local unlocked account, the transaction will be signed locally.

5.46.1 Parameters

1. `Object` - The transaction data to sign *web3.eth.sendTransaction()* for more. 1. `string` - The address of the account. 3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.46.2 Returns

`Promise<object>` - The RLP encoded transaction. The `raw` property can be used to send the transaction using *web3.eth.sendSignedTransaction*.

5.46.3 Example

```
web3.eth.signTransaction({
    from: "0xEB014f8c8B418Db6b45774c326A0E64C78914dC0",
    gasPrice: "20000000000",
    gas: "21000",
    to: '0x3535353535353535353535353535353535353535',
    value: "1000000000000000000",
    data: ""
}).then(console.log);
> {
    raw:
↪'0xf86c808504a817c800825208943535353535353535353535353535353535353535880de0b6b3a76400008025a04f4c1
↪',
    tx: {
        nonce: '0x0',
        gasPrice: '0x4a817c800',
        gas: '0x5208',
        to: '0x3535353535353535353535353535353535353535',
        value: '0xde0b6b3a7640000',
        input: '0x',
        v: '0x25',
        r: '0x4f4c17305743700648bc4f6cd3038ec6f6af0df73e31757007b7f59df7bee88d',
        s: '0x7e1941b264348e80c78c4027afc65a87b0a5e43e86742b8ca0823584c6788fd0',
        hash: '0xda3be87732110de6c1354c83770aae630ede9ac308d9f7b399ecfba23d923384'
    }
}
```

5.47 call

```
web3.eth.call(callObject [, defaultBlock] [, callback])
```

Executes a message call transaction, which is directly executed in the VM of the node, but never mined into the blockchain.

5.47.1 Parameters

1. `Object` - A transaction object see *web3.eth.sendTransaction*, with the difference that for calls the `from` property is optional as well.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with web3.eth.defaultBlock.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.47.2 Returns

`Promise<string>` - The returned data of the call, e.g. a smart contract functions return value.

5.47.3 Example

```
web3.eth.call({
    to: "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe", // contract address
    data: "0xc6888fa1000000000000000000000000000000000000000000000000000000000000000003"
}).then(console.log);
> "0x000000000000000000000000000000000000000000000000000000000000000a"
```

5.48 estimateGas

```
web3.eth.estimateGas(callObject [, callback])
```

Executes a message call or transaction and returns the amount of the gas used.

5.48.1 Parameters

1. `Object` - A transaction object see *web3.eth.sendTransaction*, with the difference that for calls the `from` property is optional as well.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.48.2 Returns

`Promise<number>` - The used gas for the simulated call/transaction.

5.48.3 Example

```
web3.eth.estimateGas({
    to: "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
    data: "0xc6888fa1000000000000000000000000000000000000000000000000000000000000000003"
}).then(console.log);
> "0x0000000000000000000000000000000000000000000000000000000000000015"
```

5.49 getPastLogs

```
web3.eth.getPastLogs(options [, callback])
```

Gets past logs, matching the given options.

5.49.1 Parameters

1. `Object` - The filter options as follows:

- `fromBlock` - `Number|String`: The number of the earliest block (`"latest"` may be given to mean the most recent and `"pending"` currently mining, block). By default `"latest"`.
- `toBlock` - `Number|String`: The number of the latest block (`"latest"` may be given to mean the most recent and `"pending"` currently mining, block). By default `"latest"`.
- `address` - `String|Array`: An address or a list of addresses to only get logs from particular account(s).
- `topics` - `Array`: An array of values which must each appear in the log entries. The order is important, if you want to leave topics out use `null`, e.g. `[null, '0x12...']`. You can also pass an array for each topic with options for that topic e.g. `[null, ['option1', 'option2']]`

5.49.2 Returns

`Promise<Array>` - Array of log objects.

The structure of the returned event `Object` in the `Array` looks as follows:

- `address` - `String`: From which this event originated from.
- `data` - `String`: The data containing non-indexed log parameter.
- `topics` - `Array`: An array with max 4 32 Byte topics, topic 1-3 contains indexed parameters of the log.
- `logIndex` - `Number`: Integer of the event index position in the block.
- `transactionIndex` - `Number`: Integer of the transaction's index position, the event was created in.
- `transactionHash` 32 Bytes - `String`: Hash of the transaction this event was created in.
- `blockHash` 32 Bytes - `String`: Hash of the block where this event was created in. `null` when its still pending.
- `blockNumber` - `Number`: The block number where this log was created in. `null` when still pending.

5.49.3 Example

```
web3.eth.getPastLogs({
    address: "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
    topics: ["0x033456732123ffff2342342dd12342434324234234fd234fd23fd4f23d4234"]
}).then(console.log);
> [{
    data: '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    topics: ['0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385']
    logIndex: 0,
    transactionIndex: 0,
    transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    blockHash: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    blockNumber: 1234,
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
},{...}]
```

5.50 getWork

```
web3.eth.getWork([callback])
```

Gets work for miners to mine on. Returns the hash of the current block, the seedHash, and the boundary condition to be met ("target").

5.50.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.50.2 Returns

`Promise<Array>` - The mining work with the following structure:

- `String` 32 Bytes - at **index 0**: current block header pow-hash
- `String` 32 Bytes - at **index 1**: the seed hash used for the DAG.
- `String` 32 Bytes - at **index 2**: the boundary condition ("target"), 2^256 / difficulty.

5.50.3 Example

```
web3.eth.getWork().then(console.log);
> [
  "0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
  "0x5EED0000000000000000000000000005EED0000000000000000000000000000",
  "0xd1ff1c01710000000000000000000000d1ff1c01710000000000000000000000"
]
```

5.51 submitWork

```
web3.eth.submitWork(nonce, powHash, digest, [callback])
```

Used for submitting a proof-of-work solution.

5.51.1 Parameters

1. `String` 8 Bytes: The nonce found (64 bits)
2. `String` 32 Bytes: The header's pow-hash (256 bits)
3. `String` 32 Bytes: The mix digest (256 bits)
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.51.2 Returns

`Promise<boolean>` - Returns `true` if the provided solution is valid, otherwise false.

5.51.3 Example

```
web3.eth.submitWork([
    "0x0000000000000001",
    "0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
    "0xD1FE5700000000000000000000000000D1FE5700000000000000000000000000"
])
.then(console.log);
> true
```

5.52 requestAccounts

```
web3.eth.requestAccounts([callback])
```

This method will request/enable the accounts from the current environment it is running (Metamask, Status or Mist). It doesn't work if you're connected to a node with a default Web3.js provider. (WebsocketProvider, HttpProvidder and IpcProvider) This method will only work if you're using the injected provider from a application like Status, Mist or Metamask.

For further information about the behavior of this method please read the EIP of it: EIP-1102

5.52.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.52.2 Returns

`Promise<Array>` - Returns an array of enabled accounts.

5.52.3 Example

```
web3.eth.requestAccounts().then(console.log);
> ['0aae0B295369a9FD31d5F28D9Ec85E40f4cb692BAf',␣
↪0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe]
```

5.53 getChainId

```
web3.eth.getChainId([callback])
```

Returns the chain ID of the current connected node as described in the EIP-695.

5.53.1 Returns

`Promise<Number>` - Returns chain ID.

5.53.2 Example

```
web3.eth.getChainId().then(console.log);
> 61
```

5.54 getNodeInfo

```
web3.eth.getNodeInfo([callback])
```

5.54.1 Returns

`Promise<String>` - The current client version.

5.54.2 Example

```
web3.eth.getNodeInfo().then(console.log);
> "Mist/v0.9.3/darwin/go1.4.1"
```

5.55 getProof

```
web3.eth.getProof(address, storageKey, blockNumber, [callback])
```

Returns the account and storage-values of the specified account including the Merkle-proof as described in EIP-1186.

5.55.1 Parameters

1. `String` 20 Bytes: The Address of the account or contract.
2. `Array` 32 Bytes: Array of storage-keys which should be proofed and included. See *web3.eth.getStorageAt*.
3. `Number | String | "latest" | "earliest"`: Integer block number, or the string "latest" or "earliest".
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.55.2 Returns

`Promise<Object>` - A account object.

> `balance` - The balance of the account. See *web3.eth.getBalance*. `codeHash` - hash of the code of the account. For a simple Account without code it will return "0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470" `nonce` - Nonce of the account. `storageHash` - SHA3 of the StorageRoot. All storage will deliver a MerkleProof starting with this rootHash. `accountProof` - Array of rlp-serialized MerkleTree-Nodes, starting with the stateRoot-Node, following the path of the SHA3 (address) as key. `storageProof` - Array of storage-entries as requested. `key` - The requested storage key. `value` - The storage value.

5.55.3 Example

```
web3.eth.getProof(
    "0x1234567890123456789012345678901234567890",
    ["0x0000000000000000000000000000000000000000000000000000000000000000",
↪"0x0000000000000000000000000000000000000000000000000000000000000001"],
    "latest"
).then(console.log);
> {
    "address": "0x1234567890123456789012345678901234567890",
    "accountProof": [

↪"0xf90211a090dcaf88c40c7bbc95a912cbdde67c175767b31173df9ee4b0d733bfdd511c43a0babe369f6b12092f49181a
↪",

↪"0xf90211a0395d87a95873cd98c21cf1df9421af03f7247880a2554e20738eec2c7507a494a0bcf6546339a1e7e14eb8fb
↪",

↪"0xf90171a04ad705ea7bf04339fa36b124fa221379bd5a38ffe9a6112cb2d94be3a437b879a08e45b5f72e8149c01efcb7
↪",

↪"0xf851808080a009833150c367df138f1538689984b8a84fc55692d3d41fe4d1e5720ff5483a6980808080808080808080
↪"
        ],
        "balance": 0,
        "codeHash":
↪"0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470",
        "nonce": 0,
        "storageHash":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
        "storageProof": [
        {
            "key": "0x0000000000000000000000000000000000000000000000000000000000000000
↪",
            "value": '0',
            "proof": []
        },
        {
            "key": "0x0000000000000000000000000000000000000000000000000000000000000001
↪",
            "value": '0',
            "proof": []
        }
    ]
}
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 6

web3.eth.subscribe

The `web3.eth.subscribe` function lets you subscribe to specific events in the blockchain.

6.1 subscribe

```
web3.eth.subscribe(type [, options] [, callback]);
```

6.1.1 Parameters

1. `String` - The subscription, you want to subscribe to.
2. `Mixed` - (optional) Optional additional parameters, depending on the subscription type.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second. Will be called for each incoming subscription, and the subscription itself as 3 parameter.

6.1.2 Returns

`EventEmitter` - A Subscription instance

- `subscription.id`: The subscription id, used to identify and unsubscribing the subscription.
- `subscription.subscribe([callback])`: Can be used to re-subscribe with the same parameters.
- `subscription.unsubscribe([callback])`: Unsubscribes the subscription and returns *TRUE* in the callback if successfull.
- `subscription.options`: The subscription options, used when re-subscribing.
- `subscription.type`: The subscription type.
- `subscription.method`: The subscription method e.g.: `logs`.
- `on("data")` returns `Object`: Fires on each incoming log with the log object as argument.

- `on("changed")` returns `Object`: Fires on each log which was removed from the blockchain. The log will have the additional property `"removed: true"`.
- `on("error")` returns `Object`: Fires when an error in the subscription occurs.

6.1.3 Notification returns

- `any` - depends on the subscription, see the different subscriptions for more.

6.1.4 Example

```
const subscription = web3.eth.subscribe('logs', {
    address: '0x123456..',
    topics: ['0x12345...']
}, function(error, result){
    if (!error)
        console.log(result);
});

// unsubscribes the subscription
subscription.unsubscribe(function(error, success){
    if(success)
        console.log('Successfully unsubscribed!');
});
```

6.2 clearSubscriptions

```
web3.eth.clearSubscriptions()
```

Resets subscriptions.

Note: This will not reset subscriptions from other packages like `web3-shh`.

6.2.1 Returns

`Promise<boolean>`

6.2.2 Example

```
web3.eth.subscribe('logs', {} ,function(){ ... });

...

web3.eth.clearSubscriptions();
```

6.3 subscribe("pendingTransactions")

```
web3.eth.subscribe('pendingTransactions' [, callback]);
```

Subscribes to incoming pending transactions.

6.3.1 Parameters

1. `String` - `"pendingTransactions"`, the type of the subscription.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second. Will be called for each incoming subscription.

6.3.2 Returns

`EventEmitter`: An *subscription instance* as an event emitter with the following events:

- `"data"` returns `String`: Fires on each incoming pending transaction and returns the transaction hash.
- `"error"` returns `Object`: Fires when an error in the subscription occurs.

6.3.3 Notification returns

1. `Object|Null` - First parameter is an error object if the subscription failed.
2. `String` - Second parameter is the transaction hash.

6.3.4 Example

```
const subscription = web3.eth.subscribe('pendingTransactions', function(error, result)
↪{
    if (!error)
        console.log(result);
})
.on("data", function(transaction){
    console.log(transaction);
});

// unsubscribes the subscription
subscription.unsubscribe(function(error, success){
    if(success)
        console.log('Successfully unsubscribed!');
});
```

6.4 subscribe("newBlockHeaders")

```
web3.eth.subscribe('newBlockHeaders' [, callback]);
```

Subscribes to incoming block headers. This can be used as timer to check for changes on the blockchain.

6.4.1 Parameters

1. `String` - `"newBlockHeaders"`, the type of the subscription.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second. Will be called for each incoming subscription.

6.4.2 Returns

`EventEmitter`: An *subscription instance* as an event emitter with the following events:

- `"data"` returns `Object`: Fires on each incoming block header.
- `"error"` returns `Object`: Fires when an error in the subscription occurs.

The structure of a returned block header is as follows:

- `number` - `Number`: The block number. `null` when its pending block.
- `hash` 32 Bytes - `String`: Hash of the block. `null` when its pending block.
- `parentHash` 32 Bytes - `String`: Hash of the parent block.
- `nonce` 8 Bytes - `String`: Hash of the generated proof-of-work. `null` when its pending block.
- `sha3Uncles` 32 Bytes - `String`: SHA3 of the uncles data in the block.
- `logsBloom` 256 Bytes - `String`: The bloom filter for the logs of the block. `null` when its pending block.
- `transactionsRoot` 32 Bytes - `String`: The root of the transaction trie of the block
- `stateRoot` 32 Bytes - `String`: The root of the final state trie of the block.
- `receiptsRoot` 32 Bytes - `String`: Transaction receipts are used to store the state after a transaction has been executed and are kept in an index-keyed trie. The hash of its root is placed in the block header as the receipts root.
- `miner` - `String`: The address of the beneficiary to whom the mining rewards were given.
- `extraData` - `String`: The "extra data" field of this block.
- `gasLimit` - `Number`: The maximum gas allowed in this block.
- `gasUsed` - `Number`: The total used gas by all transactions in this block. It can be multiplied to *gasPrice* to obtain total amount in wei.
- `timestamp` - `Number`: The unix timestamp for when the block was collated.

6.4.3 Notification returns

1. `Object|Null` - First parameter is an error object if the subscription failed.
2. `Object` - The block header object like above.

6.4.4 Example

```
const subscription = web3.eth.subscribe('newBlockHeaders', function(error, result){
    if (!error) {
        console.log(result);
```

(continues on next page)

(continued from previous page)

```
        return;
    }

    console.error(error);
})
.on("data", function(blockHeader){
    console.log(blockHeader);
})
.on("error", console.error);

// unsubscribes the subscription
subscription.unsubscribe(function(error, success){
    if (success) {
        console.log('Successfully unsubscribed!');
    }
});
```

6.5 subscribe("syncing")

```
web3.eth.subscribe('syncing' [, callback]);
```

Subscribe to syncing events. This will return an object when the node is syncing and when its finished syncing will return `FALSE`.

6.5.1 Parameters

1. `String` - `"syncing"`, the type of the subscription.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second. Will be called for each incoming subscription.

6.5.2 Returns

`EventEmitter`: An *subscription instance* as an event emitter with the following events:

- `"data"` returns `Object`: Fires on each incoming sync object as argument.
- `"changed"` returns `Object`: Fires when the synchronisation is started with `true` and when finished with `false`.
- `"error"` returns `Object`: Fires when an error in the subscription occurs.

For the structure of a returned event `Object` see *web3.eth.isSyncing return values*.

6.5.3 Notification returns

1. `Object|Null` - First parameter is an error object if the subscription failed.
2. `Object|Boolean` - The syncing object, when started it will return `true` once or when finished it will return *false* once.

6.5.4 Example

```
const subscription = web3.eth.subscribe('syncing', function(error, sync){
    if (!error)
        console.log(sync);
})
.on("data", function(sync){
    // show some syncing stats
})
.on("changed", function(isSyncing){
    if(isSyncing) {
        // stop app operation
    } else {
        // regain app operation
    }
});

// unsubscribes the subscription
subscription.unsubscribe(function(error, success){
    if(success)
        console.log('Successfully unsubscribed!');
});
```

6.6 subscribe("logs")

```
web3.eth.subscribe('logs', options [, callback]);
```

Subscribes to incoming logs, filtered by the given options.

6.6.1 Parameters

1. `"logs"` - `String`, the type of the subscription.
2. `Object` - The subscription options
   - `fromBlock` - `Number`: The number of the earliest block. By default `null`.
   - `address` - `String|Array`: An address or a list of addresses to only get logs from particular account(s).
   - `topics` - `Array`: An array of values which must each appear in the log entries. The order is important, if you want to leave topics out use `null`, e.g. `[null, '0x00...']`. You can also pass another array for each topic with options for that topic e.g. `[null, ['option1', 'option2']]`
3. `callback` - `Function`: (optional) Optional callback, returns an error object as first parameter and the result as second. Will be called for each incoming subscription.

6.6.2 Returns

`EventEmitter`: An *subscription instance* as an event emitter with the following events:

- `"data"` returns `Object`: Fires on each incoming log with the log object as argument.
- `"changed"` returns `Object`: Fires on each log which was removed from the blockchain. The log will have the additional property `"removed: true"`.

- `"error"` returns `Object`: Fires when an error in the subscription occurs.

For the structure of a returned event `Object` see *web3.eth.getPastEvents return values*.

6.6.3 Notification returns

1. `Object|Null` - First parameter is an error object if the subscription failed.
2. `Object` - The log object like in *web3.eth.getPastEvents return values*.

6.6.4 Example

```
const subscription = web3.eth.subscribe('logs', {
    address: '0x123456..',
    topics: ['0x12345...']
}, (error, result) => {
    if (!error) {
        console.log(result);
    }

    console.error(error);
})
.on("data", (log) => {
    console.log(log);
})
.on("changed", (log) => {
    console.log(log);
});

// unsubscribes the subscription
subscription.unsubscribe((error, success) => {
    if (success) {
        console.log('Successfully unsubscribed!');
    }
});
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 7

web3.eth.Contract

The `web3.eth.Contract` object makes it easy to interact with smart contracts on the Ethereum blockchain. When you create a new contract object you give it the json interface of the respective smart contract and web3 will auto convert all calls into low level ABI calls over RPC for you.

This allows you to interact with smart contracts as if they were JavaScript objects.

To use it standalone:

7.1 web3.eth.Contract

```
new web3.eth.Contract(jsonInterface, address, options)
```

Creates a new contract instance with all its methods and events defined in its json interface object.

7.1.1 Parameters

1. `jsonInterface` - `Array`: The json interface for the contract to instantiate
2. `address` - `String` (optional): This address is necessary for transactions and call requests and can also be added later using `myContract.options.address = '0x1234..'`.
3. **`options` - `Object` (optional): The options of the contract. Some are used as fallbacks for calls and transactions:**

    - `data` - `String`: The byte code of the contract. Used when the contract gets *deployed*.
    - `address` - `String`: The address where the contract is deployed. See *address*.
    - *defaultAccount*
    - *defaultBlock*
    - *defaultGas*

- *defaultGasPrice*
- *transactionBlockTimeout*
- *transactionConfirmationBlocks*
- *transactionPollingTimeout*
- *transactionSigner*

7.1.2 Returns

`Object`: The contract instance with all its methods and events.

7.1.3 Example

```
const myContract = new web3.eth.Contract([...],
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe', {
    defaultAccount: '0x1234567890123456789012345678901234567891', // default from␣
↪address
    defaultGasPrice: '20000000000' // default gas price in wei, 20 gwei in this case
});
```

7.2 = Properties =

7.3 options

The contract options object has the following properties:

- `data` - `String`: The contract bytecode.
- `address` - `String` (deprecated use `contract.address`): The address of the contract.

7.4 address

```
myContract.address
```

The address used for this contract instance. All transactions generated by web3.js from this contract will contain this address as the "to".

The address will be stored in lowercase.

7.4.1 Property

`address` - `String|null`: The address for this contract, or `null` if it's not yet set.

7.4.2 Example

```
myContract.address;
> '0xde0b295669a9fd93d5f28d9ec85e40f4cb697bae'

// set a new address
myContract.address = '0x1234FFDD...';
```

7.5 jsonInterface

```
myContract.jsonInterface
```

The json interface object derived from the ABI of this contract.

7.5.1 Property

`jsonInterface` - `AbiModel`: The json interface for this contract. Re-setting this will regenerate the methods and events of the contract instance.

7.5.2 AbiModel

```
interface AbiModel {
    getMethod(name: string): AbiItemModel | false;
    getMethods(): AbiItemModel[];
    hasMethod(name: string): boolean;
    getEvent(name: string): AbiItemModel | false;
    getEvents(): AbiItemModel[];
    getEventBySignature(signature: string): AbiItemModel;
    hasEvent(name: string): boolean;
}

interface AbiItemModel {
    name: string;
    signature: string;
    payable: boolean;
    anonymous: boolean;
    getInputLength(): Number;
    getInputs(): AbiInput[];
    getIndexedInputs(): AbiInput[];
    getOutputs(): AbiOutput[];
    isOfType(): boolean;
}

interface AbiInput {
    name: string;
    type: string;
    indexed?: boolean;
    components?: AbiInput[];
}
```

(continues on next page)

(continued from previous page)

```
interface AbiOutput {
    name: string;
    type: string;
    components?: AbiOutput[];
}
```

7.6 = Methods =

7.7 clone

```
myContract.clone()
```

Clones the current contract instance.

7.7.1 Parameters

none

7.7.2 Returns

`Object`: The new contract instance.

7.7.3 Example

```
const contract1 = new eth.Contract(abi, address, {gasPrice: '12345678',␣
↪defaultAccount: fromAddress});

const contract2 = contract1.clone();
contract2.address = address2;

(contract1.address !== contract2.address);
> true
```

7.8 deploy

```
myContract.deploy(options)
```

Call this function to deploy the contract to the blockchain. After successful deployment the promise will resolve with a new contract instance.

7.8.1 Parameters

1. **`options` - `Object`: The options used for deployment.**
    - `data` - `String`: The byte code of the contract.
    - `arguments` - `Array` (optional): The arguments which get passed to the constructor on deployment.

7.8.2 Returns

`Object`: The transaction object:

- `Array` - arguments: The arguments passed to the method before. They can be changed.
- `Function` - *send*: Will deploy the contract. The promise will resolve with the new contract instance, instead of the receipt!
- `Function` - *estimateGas*: Will estimate the gas used for deploying.
- `Function` - *encodeABI*: Encodes the ABI of the deployment, which is contract data + constructor parameters

For details to the methods see the documentation below.

7.8.3 Example

```
myContract.deploy({
    data: '0x12345...',
    arguments: [123, 'My String']
})
.send({
    from: '0x1234567890123456789012345678901234567891',
    gas: 1500000,
    gasPrice: '30000000000000'
}, (error, transactionHash) => { ... })
.on('error', (error) => { ... })
.on('transactionHash', (transactionHash) => { ... })
.on('receipt', (receipt) => {
   console.log(receipt.contractAddress) // contains the new contract address
})
.on('confirmation', (confirmationNumber, receipt) => { ... })
.then((newContractInstance) => {
    console.log(newContractInstance.options.address) // instance with the new␣
↪contract address
});


// When the data is already set as an option to the contract itself
myContract.options.data = '0x12345...';

myContract.deploy({
    arguments: [123, 'My String']
})
.send({
    from: '0x1234567890123456789012345678901234567891',
    gas: 1500000,
    gasPrice: '30000000000000'
})
```

(continues on next page)

(continued from previous page)

```
.then((newContractInstance) => {
    console.log(newContractInstance.options.address) // instance with the new␣
↪contract address
});


// Simply encoding
myContract.deploy({
    data: '0x12345...',
    arguments: [123, 'My String']
})
.encodeABI();
> '0x12345...0000012345678765432'


// Gas estimation
myContract.deploy({
    data: '0x12345...',
    arguments: [123, 'My String']
})
.estimateGas((err, gas) => {
    console.log(gas);
});
```

7.9 methods

```
myContract.methods.myMethod([param1[, param2[, ...]]])
```

Creates a transaction object for that method, which then can be *called*, *send*, estimated.

The methods of this smart contract are available through:

- The name: `myContract.methods.myMethod(123)`
- The name with parameters: `myContract.methods['myMethod(uint256)'](123)`
- The signature: `myContract.methods['0x58cf5f10'](123)`

This allows calling functions with same name but different parameters from the JavaScript contract object.

7.9.1 Parameters

Parameters of any method depend on the smart contracts methods, defined in the JSON interface.

7.9.2 Returns

`Object`: The Transaction Object:

- `Array` - arguments: The arguments passed to the method before. They can be changed.
- `Function` - *call*: Will call the "constant" method and execute its smart contract method in the EVM without sending a transaction (Can't alter the smart contract state).

- `Function` - *send*: Will send a transaction to the smart contract and execute its method (Can alter the smart contract state).
- `Function` - *estimateGas*: Will estimate the gas used when the method would be executed on chain.
- `Function` - *encodeABI*: Encodes the ABI for this method. This can be send using a transaction, call the method or passing into another smart contracts method as argument.

  For details to the methods see the documentation below.

7.9.3 Example

```
// calling a method

myContract.methods.myMethod(123).call({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'}, (error, result) => {
    ...
});

// or sending and using a promise
myContract.methods.myMethod(123).send({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
.then((receipt) => {
    // receipt can also be a new contract instance, when coming from a "contract.
↪deploy({...}).send()"
});

// or sending and using the events

myContract.methods.myMethod(123).send({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
.on('transactionHash', (hash) => {
    ...
})
.on('receipt', (receipt) => {
    ...
})
.on('confirmation', (confirmationNumber, receipt) => {
    ...
})
.on('error', console.error);
```

7.10 methods.myMethod.call

```
myContract.methods.myMethod([param1[, param2[, ...]]]).call(transactionObject,␣
↪blockNumber, callback])
```

Will call a "constant" method and execute its smart contract method in the EVM without sending any transaction. Note calling can not alter the smart contract state.

7.10.1 Parameters

`options` - `Object` (optional): The options used for calling. 1.* `transactionObject`

- `from` - `String` (optional): The address the call "transaction" should be made from.
- `gasPrice` - `String` (optional): The gas price in wei to use for this call "transaction".It is the wei per unit of gas.
- `gas` - `Number` (optional): The maximum gas provided for this call "transaction" (gas limit).

2.*‘‘blockNumber‘‘ - `Number`: The block number this log was created in. `null` when still pending. 3.*‘‘callback‘‘ - `Function` (optional): This callback will be fired with the result of the smart contract method execution as the second argument, or with an error object as the first argument.

7.10.2 Returns

`Promise<any>` - The return value(s) of the smart contract method. If it returns a single value, it's returned as is. If it has multiple return values they are returned as an object with properties and indices:

7.10.3 Example

```
// using the callback
myContract.methods.myMethod(123).call({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'}, (error, result) => {
    ...
});

// using the promise
myContract.methods.myMethod(123).call({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
.then((result) => {
    ...
});


// MULTI-ARGUMENT RETURN:

// Solidity
contract MyContract {
    function myFunction() returns(uint256 myNumber, string myString) {
        return (23456, "Hello!%");
    }
}

// web3.js
const MyContract = new web3.eth.Contract(abi, address);
MyContract.methods.myFunction().call()
.then(console.log);
> Result {
    myNumber: '23456',
    myString: 'Hello!%',
    0: '23456', // these are here as fallbacks if the name is not know or given
    1: 'Hello!%'
}


// SINGLE-ARGUMENT RETURN:

// Solidity
```

(continues on next page)

(continued from previous page)

```
contract MyContract {
    function myFunction() returns(string myString) {
        return "Hello!%";
    }
}

// web3.js
const MyContract = new web3.eth.Contract(abi, address);
MyContract.methods.myFunction().call()
.then(console.log);
> "Hello!%"
```

7.11 methods.myMethod.send

```
myContract.methods.myMethod([param1[, param2[, ...]]]).send(options[, callback])
```

Will send a transaction to the smart contract and execute its method. Note this can alter the smart contract state.

7.11.1 Parameters

1. **`options` - `Object`: The options used for sending.**
    - `from` - `String`: The address the transaction should be sent from.
    - `gasPrice` - `String` (optional): The gas price in wei to use for this transaction.It is the wei per unit of gas.
    - `gas` - `Number` (optional): The maximum gas provided for this transaction (gas limit).
    - `value` - ‘‘Number|String|BN|BigNumber‘‘(optional): The value transferred for the transaction in wei.
2. `callback` - `Function` (optional): This callback will be fired first with the “transactionHash”, or with an error object as the first argument.

7.11.2 Returns

The **callback** will return the 32 bytes transaction hash.

`PromiEvent`: A *promise combined event emitter*. Will be resolved when the transaction *receipt* is available, OR if this `send()` is called from a `someContract.deploy()`, then the promise will resolve with the *new contract instance*. Additionally the following events are available:

- `"transactionHash"` returns `String`: is fired right after the transaction is sent and a transaction hash is available.
- `"receipt"` returns `Object`: is fired when the transaction *receipt* is available. Receipts from contracts will have no `logs` property, but instead an `events` property with event names as keys and events as properties. See *getPastEvents return values* for details about the returned event object.
- `"confirmation"` returns `Number`, `Object`: is fired for every confirmation up to the 24th confirmation. Receives the confirmation number as the first and the receipt as the second argument. Fired from confirmation 1 on, which is the block where it’s mined.

- "error" returns Error: is fired if an error occurs during sending. If an out of gas error, the second parameter is the receipt.

7.11.3 Example

```
// using the callback
myContract.methods.myMethod(123).send({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'}, (error, transactionHash) => {
    ...
});

// using the promise
myContract.methods.myMethod(123).send({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
.then((receipt) => {
    // receipt can also be a new contract instance, when coming from a "contract.
↪deploy({...}).send()"
});


// using the event emitter
myContract.methods.myMethod(123).send({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
.on('transactionHash', (hash) => {
    ...
})
.on('confirmation', (confirmationNumber, receipt) => {
    ...
})
.on('receipt', (receipt) => {
    // receipt example
    console.log(receipt);
    > {
        "transactionHash":
↪"0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
        "transactionIndex": 0,
        "blockHash":
↪"0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
        "blockNumber": 3,
        "contractAddress": "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
        "cumulativeGasUsed": 314159,
        "gasUsed": 30234,
        "events": {
            "MyEvent": {
                returnValues: {
                    myIndexedParam: 20,
                    myOtherIndexedParam: '0x123456789...',
                    myNonIndexParam: 'My String'
                },
                raw: {
                    data:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
                    topics: [
↪'0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385']
                },
```

(continues on next page)

(continued from previous page)

```
                event: 'MyEvent',
                signature:
↪'0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
                logIndex: 0,
                transactionIndex: 0,
                transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
                blockHash:
↪'0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
                blockNumber: 1234,
                address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
            },
            "MyOtherEvent": {
                ...
            },
            "MyMultipleEvent":[{...}, {...}] // If there are multiple of the same␣
↪event, they will be in an array
        }
    }
})
.on('error', console.error); // If there's an out of gas error the second parameter␣
↪is the receipt.
```

7.12 methods.myMethod.estimateGas

```
myContract.methods.myMethod([param1[, param2[, ...]]]).estimateGas(options[,␣
↪callback])
```

Will call estimate the gas a method execution will take when executed in the EVM without sending a transaction. The estimation can differ from the actual gas used when later sending a transaction, as the state of the smart contract can be different at that time.

7.12.1 Parameters

1. **`options` - `Object` (optional): The options used for calling.**
    - `from` - `String` (optional): The address the call "transaction" should be made from.
    - `gas` - `Number` (optional): The maximum gas provided for this call "transaction" (gas limit). Setting a specific value helps to detect out of gas errors. If all gas is used it will return the same number.
    - `value` - ‘‘Number|String|BN|BigNumber‘‘(optional): The value transferred for the call "transaction" in wei.
2. `callback` - `Function` (optional): This callback will be fired with the result of the gas estimation as the second argument, or with an error object as the first argument.

7.12.2 Returns

`Promise<number>` - The gas amount estimated.

7.12.3 Example

```
// using the callback
myContract.methods.myMethod(123).estimateGas({gas: 5000000}, function(error,␣
↪gasAmount){
    if(gasAmount == 5000000)
        console.log('Method ran out of gas');
});

// using the promise
myContract.methods.myMethod(123).estimateGas({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
.then(function(gasAmount){
    ...
})
.catch(function(error){
    ...
});
```

7.13 methods.myMethod.encodeABI

```
myContract.methods.myMethod([param1[, param2[, ...]]]).encodeABI()
```

Encodes the ABI for this method. This can be used to send a transaction, call a method, or pass it into another smart contracts method as arguments.

7.13.1 Parameters

none

7.13.2 Returns

`String`: The encoded ABI byte code to send via a transaction or call.

7.13.3 Example

```
myContract.methods.myMethod(123).encodeABI();
> '0x58cf5f1000000000000000000000000000000000000000000000000000000000000007B'
```

7.14 = Events =

7.15 once

```
myContract.once(event[, options], callback)
```

Subscribes to an event and unsubscribes immediately after the first event or error. Will only fire for a single event.

7.15.1 Parameters

1. `event` - `String`: The name of the event in the contract, or `"allEvents"` to get all events.
2. **`options` - `Object` (optional): The options used for deployment.**
    - `filter` - `Object` (optional): Lets you filter events by indexed parameters, e.g. `{filter: {myNumber: [12,13]}}` means all events where "myNumber" is 12 or 13.
    - `topics` - `Array` (optional): This allows you to manually set the topics for the event filter. If given the filter property and event signature, (topic[0]) will not be set automatically.
3. `callback` - `Function`: This callback will be fired for the first *event* as the second argument, or an error as the first argument. See *getPastEvents return values* for details about the event structure.

7.15.2 Returns

`undefined`

7.15.3 Example

```
myContract.once('MyEvent', {
    filter: {myIndexedParam: [20,23], myOtherIndexedParam: '0x123456789...'}, //␣
↪Using an array means OR: e.g. 20 or 23
    fromBlock: 0
}, (error, event) => { console.log(event); });

// event output example
> {
    returnValues: {
        myIndexedParam: 20,
        myOtherIndexedParam: '0x123456789...',
        myNonIndexParam: 'My String'
    },
    raw: {
        data: '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
        topics: ['0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7
↪', '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385']
    },
    event: 'MyEvent',
    signature: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    logIndex: 0,
    transactionIndex: 0,
    transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    blockHash: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    blockNumber: 1234,
```

(continues on next page)

(continued from previous page)

```
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
}
```

7.16 events

```
myContract.events.MyEvent([options][, callback])
```

Subscribe to an event

7.16.1 Parameters

1. **`options` - `Object` (optional): The options used for deployment.**
    - `filter` - `Object` (optional): Let you filter events by indexed parameters, e.g. `{filter: {myNumber: [12,13]}}` means all events where "myNumber" is 12 or 13.
    - `fromBlock` - `Number` (optional): The block number from which to get events on.
    - `topics` - `Array` (optional): This allows to manually set the topics for the event filter. If given the filter property and event signature, (topic[0]) will not be set automatically.
2. `callback` - `Function` (optional): This callback will be fired for each *event* as the second argument, or an error as the first argument.

7.16.2 Returns

`EventEmitter`: The event emitter has the following events:

- `"data"` returns `Object`: Fires on each incoming event with the event object as argument.
- `"changed"` returns `Object`: Fires on each event which was removed from the blockchain. The event will have the additional property `"removed: true"`.
- `"error"` returns `Object`: Fires when an error in the subscription occurs.

The structure of the returned event `Object` looks as follows:

- `event` - `String`: The event name.
- `signature` - `String|Null`: The event signature, `null` if it's an anonymous event.
- `address` - `String`: Address this event originated from.
- `returnValues` - `Object`: The return values coming from the event, e.g. `{myVar: 1, myVar2: '0x234...'}`.
- `logIndex` - `Number`: Integer of the event index position in the block.
- `transactionIndex` - `Number`: Integer of the transaction's index position the event was created in.
- `transactionHash` 32 Bytes - `String`: Hash of the transaction this event was created in.
- `blockHash` 32 Bytes - `String`: Hash of the block this event was created in. `null` when it's still pending.
- `blockNumber` - `Number`: The block number this log was created in. `null` when still pending.

- `raw.data` - `String`: The data containing non-indexed log parameter.
- `raw.topics` - `Array`: An array with max 4 32 Byte topics, topic 1-3 contains indexed parameters of the event.

7.16.3 Example

```
myContract.events.MyEvent({
    filter: {myIndexedParam: [20,23], myOtherIndexedParam: '0x123456789...'}, //␣
↪Using an array means OR: e.g. 20 or 23
    fromBlock: 0
}, (error, event) => { console.log(event); })
.on('data', (event) => {
    console.log(event); // same results as the optional callback above
})
.on('changed', (event) => {
    // remove event from local database
})
.on('error', console.error);

// event output example
> {
    returnValues: {
        myIndexedParam: 20,
        myOtherIndexedParam: '0x123456789...',
        myNonIndexParam: 'My String'
    },
    raw: {
        data: '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
        topics: ['0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7
↪', '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385']
    },
    event: 'MyEvent',
    signature: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    logIndex: 0,
    transactionIndex: 0,
    transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    blockHash: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    blockNumber: 1234,
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
}
```

7.17 events.allEvents

```
myContract.events.allEvents([options][, callback])
```

Same as *events* but receives all events from this smart contract. Optionally the filter property can filter those events.

7.18 getPastEvents

```
myContract.getPastEvents(event[, options][, callback])
```

Gets past events for this contract.

7.18.1 Parameters

1. `event` - `String`: The name of the event in the contract, or `"allEvents"` to get all events.
2. **`options` - `Object` (optional): The options used for deployment.**
    - `filter` - `Object` (optional): Lets you filter events by indexed parameters, e.g. `{filter: {myNumber: [12,13]}}` means all events where "myNumber" is 12 or 13.
    - `fromBlock` - `Number` (optional): The block number from which to get events on.
    - `toBlock` - `Number` (optional): The block number to get events up to (Defaults to `"latest"`).
    - `topics` - `Array` (optional): This allows manually setting the topics for the event filter. If given the filter property and event signature, (topic[0]) will not be set automatically.
3. `callback` - `Function` (optional): This callback will be fired with an array of event logs as the second argument, or an error as the first argument.

7.18.2 Returns

`Promise` returns `Array`: An array with the past event `Objects`, matching the given event name and filter.

For the structure of a returned event `Object` see *getPastEvents return values*.

7.18.3 Example

```
myContract.getPastEvents('MyEvent', {
    filter: {myIndexedParam: [20,23], myOtherIndexedParam: '0x123456789...'}, //␣
↪Using an array means OR: e.g. 20 or 23
    fromBlock: 0,
    toBlock: 'latest'
}, (error, events) => { console.log(events); })
.then((events) => {
    console.log(events) // same results as the optional callback above
});

> [{
    returnValues: {
        myIndexedParam: 20,
        myOtherIndexedParam: '0x123456789...',
        myNonIndexParam: 'My String'
    },
    raw: {
        data: '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
        topics: ['0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7
↪', '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385']
    },
    event: 'MyEvent',
```

(continues on next page)

(continued from previous page)

```
    signature: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    logIndex: 0,
    transactionIndex: 0,
    transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    blockHash: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    blockNumber: 1234,
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
},{
    ...
}]
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 8

web3.eth.accounts

The `web3.eth.accounts` contains functions to generate Ethereum accounts and sign transactions and data.

Note: This package got NOT audited until now. Take precautions to clear memory properly, store the private keys safely, and test transaction receiving and sending functionality properly before using in production!

```
import {Accounts} from 'web3-eth-accounts';

// Passing in the eth or web3 package is necessary to allow retrieving chainId,␣
↪gasPrice and nonce automatically
// for accounts.signTransaction().
const accounts = new Accounts('ws://localhost:8546', null, options);
```

8.1 create

```
web3.eth.accounts.create([entropy]);
```

Generates an account object with private key and public key. It's different from *web3.eth.personal.newAccount()* which creates an account over the network on the node via an RPC call.

8.1.1 Parameters

1. `entropy` - `String` (optional): A random string to increase entropy. If given it should be at least 32 characters. If none is given a random string will be generated using randomhex.

8.1.2 Returns

`Object` - The account object with the following structure:

- `address` - `string`: The account address.
- `privateKey` - `string`: The accounts private key. This should never be shared or stored unencrypted in localstorage! Also make sure to `null` the memory after usage.
- `signTransaction(tx [, callback])` - `Function`: The function to sign transactions. See *web3.eth.accounts.signTransaction()* for more.
- `sign(data)` - `Function`: The function to sign transactions. See *web3.eth.accounts.sign()* for more.

8.1.3 Example

```
web3.eth.accounts.create();
> {
    address: "0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01",
    privateKey: "0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}

web3.eth.accounts.create('2435@#@#@±±±±!!!!
↪67854321345676432 1§34567543213456785432134567');
> {
    address: "0xF2CD2AA0c7926743B1D4310b2BC984a0a453c3d4",
    privateKey: "0xd7325de5c2c1cf0009fac77d3d04a9c004b038883446b065871bc3e831dcd098",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}

web3.eth.accounts.create(web3.utils.randomHex(32));
> {
    address: "0xe78150FaCD36E8EB00291e251424a0515AA1FF05",
    privateKey: "0xcc505ee6067fba3f6fc2050643379e190e087aeffe5d958ab9f2f3ed3800fa4e",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}
```

8.2 privateKeyToAccount

```
web3.eth.accounts.privateKeyToAccount(privateKey);
```

Creates an account object from a private key.

8.2.1 Parameters

1. `privateKey` - `String`: The private key hex string beginning with `0x`.

8.2.2 Returns

`Object` - The account object with the *structure seen here*.

8.2.3 Example

```
web3.eth.accounts.privateKeyToAccount(
↪'0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709');
> {
    address: '0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01',
    privateKey: '0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709',
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}
```

8.3 signTransaction

```
web3.eth.accounts.signTransaction(tx, privateKey [, callback]);
```

Signs an Ethereum transaction with a given private key.

8.3.1 Parameters

1. **`tx` - `Object`: The transaction's properties object as follows:**
    - `nonce` - `String`: (optional) The nonce to use when signing this transaction. Default will use *web3.eth.getTransactionCount()*.
    - `chainId` - `String`: (optional) The chain id to use when signing this transaction. Default will use *web3.eth.net.getId()*.
    - `to` - `String`: (optional) The receiver of the transaction, can be empty when deploying a contract.
    - `data` - `String`: (optional) The call data of the transaction, can be empty for simple value transfers.
    - `value` - `String`: (optional) The value of the transaction in wei.
    - `gasPrice` - `String`: (optional) The gas price set by this transaction, if empty, it will use *web3.eth.gasPrice()*
    - `gas` - `String`: The gas provided by the transaction.
2. `privateKey` - `String`: The private key to sign with.
3. `callback` - `Function`: (optional) Optional callback, returns an error object as first parameter and the result as second.

8.3.2 Returns

`Promise` returning `Object`: The signed data RLP encoded transaction, or if `returnSignature` is `true` the signature value

- `messageHash` - `String`: The hash of the given message.
- `r` - `String`: First 32 bytes of the signature
- `s` - `String`: Next 32 bytes of the signature
- `v` - `String`: Recovery value + 27
- `rawTransaction` - `String`: The RLP encoded transaction, ready to be send using *web3.eth.sendSignedTransaction*.
- `transactionHash` - `String`: The transaction hash for the RLP encoded transaction.

8.3.3 Example

```
web3.eth.accounts.signTransaction({
    to: '0xF0109fC8DF283027b6285cc889F5aA624EaC1F55',
    value: '1000000000',
    gas: 2000000
}, '0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318')
.then(console.log);
> {
    messageHash: '0x88cfbd7e51c7a40540b233cf68b62ad1df3e92462f1c6018d6d67eae0f3b08f5',
    v: '0x25',
    r: '0xc9cf86333bcb065d140032ecaab5d9281bde80f21b9687b3e94161de42d51895',
    s: '0x727a108a0b8d101465414033c3f705a9c7b826e596766046ee1183dbc8aeaa68',
    rawTransaction:
↪'0xf869808504e3b29200831e848094f0109fc8df283027b6285cc889f5aa624eac1f55843b9aca008025a0c9cf86333bcb
↪',
    transactionHash:
↪'0xde8db924885b0803d2edc335f745b2b8750c8848744905684c20b987443a9593'
}

web3.eth.accounts.signTransaction({
    to: '0xF0109fC8DF283027b6285cc889F5aA624EaC1F55',
    value: '1000000000',
    gas: 2000000,
    gasPrice: '234567897654321',
    nonce: 0,
    chainId: 1
}, '0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318')
.then(console.log);
> {
    messageHash: '0x6893a6ee8df79b0f5d64a180cd1ef35d030f3e296a5361cf04d02ce720d32ec5',
    r: '0x9ebb6ca057a0535d6186462bc0b465b561c94a295bdb0621fc19208ab149a9c',
    s: '0x440ffd775ce91a833ab410777204d5341a6f9fa91216a6f3ee2c051fea6a0428',
    v: '0x25',
    rawTransaction:
↪'0xf86a8086d55698372431831e848094f0109fc8df283027b6285cc889f5aa624eac1f55843b9aca008025a009ebb6ca05
↪',
    transactionHash:
↪'0xd8f64a42b57be0d565f385378db2f6bf324ce14a594afc05de90436e9ce01f60'
}
```

8.4 recoverTransaction

```
web3.eth.accounts.recoverTransaction(rawTransaction);
```

Recovers the Ethereum address which was used to sign the given RLP encoded transaction.

8.4.1 Parameters

1. `signature` - `String`: The RLP encoded transaction.

8.4.2 Returns

`String`: The Ethereum address used to sign this transaction.

8.4.3 Example

```
web3.eth.accounts.recoverTransaction(
↪'0xf86180808401ef364594f0109fc8df283027b6285cc889f5aa624eac1f5580801ca031573280d608f75137e33fc14655
↪');
> "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55"
```

8.5 hashMessage

```
web3.eth.accounts.hashMessage(message);
```

Hashes the given message to be passed *web3.eth.accounts.recover()* function. The data will be UTF-8 HEX decoded and enveloped as follows: `"\x19Ethereum Signed Message:\n" + message.length + message` and hashed using keccak256.

8.5.1 Parameters

1. `message` - `String`: A message to hash, if its HEX it will be UTF8 decoded before.

8.5.2 Returns

`String`: The hashed message

8.5.3 Example

```
web3.eth.accounts.hashMessage("Hello World")
> "0xa1de988600a42c4b4ab089b619297c17d53cffae5d5120d82d8a92d0bb3b78f2"

// the below results in the same hash
web3.eth.accounts.hashMessage(web3.utils.utf8ToHex("Hello World"))
> "0xa1de988600a42c4b4ab089b619297c17d53cffae5d5120d82d8a92d0bb3b78f2"
```

8.6 sign

```
web3.eth.accounts.sign(data, privateKey);
```

Signs arbitrary data. This data is before UTF-8 HEX decoded and enveloped as follows: `"\x19Ethereum Signed Message:\n" + message.length + message`.

8.6.1 Parameters

1. `data` - `String`: The data to sign. If its a string it will be
2. `privateKey` - `String`: The private key to sign with.

8.6.2 Returns

`Object`: The signed data RLP encoded signature, or if `returnSignature` is `true` the signature values as follows:

- `message` - `String`: The the given message.
- `messageHash` - `String`: The hash of the given message.
- `r` - `String`: First 32 bytes of the signature
- `s` - `String`: Next 32 bytes of the signature
- `v` - `String`: Recovery value + 27

8.6.3 Example

```
web3.eth.accounts.sign('Some data',
↪'0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318');
> {
    message: 'Some data',
    messageHash: '0x1da44b586eb0729ff70a73c326926f6ed5a25f5b056e7f47fbc6e58d86871655',
    v: '0x1c',
    r: '0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd',
    s: '0x6007e74cd82e037b800186422fc2da167c747ef045e5d18a5f5d4300f8e1a029',
    signature:
↪'0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd6007e74cd82e037b800186422fc2da16
↪'
}
```

8.7 recover

```
web3.eth.accounts.recover(signatureObject);
web3.eth.accounts.recover(message, signature [, preFixed]);
web3.eth.accounts.recover(message, v, r, s [, preFixed]);
```

Recovers the Ethereum address which was used to sign the given data.

8.7.1 Parameters

1. **`message|signatureObject` - `String|Object`: Either signed message or hash, or the signature object as following**

    - `messageHash` - `String`: The hash of the given message already prefixed with `"\x19Ethereum Signed Message:\n" + message.length + message`.
    - `r` - `String`: First 32 bytes of the signature
    - `s` - `String`: Next 32 bytes of the signature
    - `v` - `String`: Recovery value + 27

2. `signature` - `String`: The raw RLP encoded signature, OR parameter 2-4 as v, r, s values.
3. `preFixed` - `Boolean` (optional, default: `false`): If the last parameter is `true`, the given message will NOT automatically be prefixed with `"\x19Ethereum Signed Message:\n" + message.length + message`, and assumed to be already prefixed.

8.7.2 Returns

`String`: The Ethereum address used to sign this data.

8.7.3 Example

```
web3.eth.accounts.recover({
    messageHash: '0x1da44b586eb0729ff70a73c326926f6ed5a25f5b056e7f47fbc6e58d86871655',
    v: '0x1c',
    r: '0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd',
    s: '0x6007e74cd82e037b800186422fc2da167c747ef045e5d18a5f5d4300f8e1a029'
})
> "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23"

// message, signature
web3.eth.accounts.recover('Some data',
↪'0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd6007e74cd82e037b800186422fc2da1
↪');
> "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23"

// message, v, r, s
web3.eth.accounts.recover('Some data', '0x1c',
↪'0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd',
↪'0x6007e74cd82e037b800186422fc2da167c747ef045e5d18a5f5d4300f8e1a029');
> "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23"
```

8.8 encrypt

```
web3.eth.accounts.encrypt(privateKey, password);
```

Encrypts a private key to the web3 keystore v3 standard.

8.8.1 Parameters

1. `privateKey` - `String`: The private key to encrypt.
2. `password` - `String`: The password used for encryption.

8.8.2 Returns

`Object`: The encrypted keystore v3 JSON.

8.8.3 Example

```
web3.eth.accounts.encrypt(
↪'0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318', 'test!')
> {
    version: 3,
    id: '04e9bcbb-96fa-497b-94d1-14df4cd20af6',
    address: '2c7536e3605d9c16a7a3d7b1898e529396a65c23',
    crypto: {
        ciphertext: 'a1c25da3ecde4e6a24f3697251dd15d6208520efc84ad97397e906e6df24d251
↪',
        cipherparams: { iv: '2885df2b63f7ef247d753c82fa20038a' },
        cipher: 'aes-128-ctr',
        kdf: 'scrypt',
        kdfparams: {
            dklen: 32,
            salt: '4531b3c174cc3ff32a6a7a85d6761b410db674807b2d216d022318ceee50be10',
            n: 262144,
            r: 8,
            p: 1
        },
        mac: 'b8b010fff37f9ae5559a352a185e86f9b9c1d7f7a9f1bd4e82a5dd35468fc7f6'
    }
}
```

8.9 decrypt

```
web3.eth.accounts.decrypt(keystoreJsonV3, password);
```

Decrypts a keystore v3 JSON, and creates the account.

8.9.1 Parameters

1. `keystoreJsonV3` - `String`: The encrypted keystore v3 JSON.
2. `password` - `String`: The password used for encryption.

8.9.2 Returns

`Object`: The decrypted account.

8.9.3 Example

```
web3.eth.accounts.decrypt({
    version: 3,
    id: '04e9bcbb-96fa-497b-94d1-14df4cd20af6',
    address: '2c7536e3605d9c16a7a3d7b1898e529396a65c23',
    crypto: {
        ciphertext: 'a1c25da3ecde4e6a24f3697251dd15d6208520efc84ad97397e906e6df24d251
↪',
        cipherparams: { iv: '2885df2b63f7ef247d753c82fa20038a' },
        cipher: 'aes-128-ctr',
        kdf: 'scrypt',
        kdfparams: {
            dklen: 32,
            salt: '4531b3c174cc3ff32a6a7a85d6761b410db674807b2d216d022318ceee50be10',
            n: 262144,
            r: 8,
            p: 1
        },
        mac: 'b8b010fff37f9ae5559a352a185e86f9b9c1d7f7a9f1bd4e82a5dd35468fc7f6'
    }
}, 'test!');
> {
    address: "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23",
    privateKey: "0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}
```

8.10 wallet

```
web3.eth.accounts.wallet;
```

Contains an in memory wallet with multiple accounts. These accounts can be used when using *web3.eth.sendTransaction()*.

8.10.1 Example

```
web3.eth.accounts.wallet;
> Wallet {
    0: {...}, // account by index
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...},  // same account by address
    "0xf0109fc8df283027b6285cc889f5aa624eac1f55": {...},  // same account by address␣
↪lowercase
    1: {...},
    "0xD0122fC8DF283027b6285cc889F5aA624EaC1d23": {...},
    "0xd0122fc8df283027b6285cc889f5aa624eac1d23": {...},

    add: function(){},
    remove: function(){},
    save: function(){},
    load: function(){},
    clear: function(){},

    length: 2,
}
```

8.11 wallet.create

```
web3.eth.accounts.wallet.create(numberOfAccounts [, entropy]);
```

Generates one or more accounts in the wallet. If wallets already exist they will not be overridden.

8.11.1 Parameters

1. `numberOfAccounts` - `Number`: Number of accounts to create. Leave empty to create an empty wallet.
2. `entropy` - `String` (optional): A string with random characters as additional entropy when generating accounts. If given it should be at least 32 characters.

8.11.2 Returns

`Object`: The wallet object.

8.11.3 Example

```
web3.eth.accounts.wallet.create(2, '54674321$3456764321$345674321$3453647544±±±$±±±!
↪!!43534534534534');
> Wallet {
    0: {...},
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...},
    "0xf0109fc8df283027b6285cc889f5aa624eac1f55": {...},
    ...
}
```

8.12 wallet.add

```
web3.eth.accounts.wallet.add(account);
```

Adds an account using a private key or account object to the wallet.

8.12.1 Parameters

1. `account` - `String|Object`: A private key or account object created with *web3.eth.accounts.create()*.

8.12.2 Returns

`Object`: The added account.

8.12.3 Example

```
web3.eth.accounts.wallet.add(
↪'0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318');
> {
    index: 0,
    address: '0x2c7536E3605D9C16a7a3D7b1898e529396a65c23',
    privateKey: '0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318',
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}

web3.eth.accounts.wallet.add({
    privateKey: '0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709',
    address: '0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01'
});
> {
    index: 0,
    address: '0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01',
    privateKey: '0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709',
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}
```

8.13 wallet.remove

```
web3.eth.accounts.wallet.remove(account);
```

Removes an account from the wallet.

8.13.1 Parameters

1. `account` - `String|Number`: The account address, or index in the wallet.

8.13.2 Returns

`Boolean`: `true` if the wallet was removed. `false` if it couldn't be found.

8.13.3 Example

```
web3.eth.accounts.wallet;
> Wallet {
    0: {...},
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...}
    1: {...},
    "0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01": {...}
    ...
}

web3.eth.accounts.wallet.remove('0xF0109fC8DF283027b6285cc889F5aA624EaC1F55');
> true

web3.eth.accounts.wallet.remove(3);
> false
```

8.14 wallet.clear

```
web3.eth.accounts.wallet.clear();
```

Securely empties the wallet and removes all its accounts.

8.14.1 Parameters

none

8.14.2 Returns

`Object`: The wallet object.

8.14.3 Example

```
web3.eth.accounts.wallet.clear();
> Wallet {
    add: function(){},
    remove: function(){},
    save: function(){},
```

(continues on next page)

(continued from previous page)

```
    load: function(){},
    clear: function(){},

    length: 0
}
```

8.15 wallet.encrypt

```
web3.eth.accounts.wallet.encrypt(password);
```

Encrypts all wallet accounts to an array of encrypted keystore v3 objects.

8.15.1 Parameters

1. `password` - `String`: The password which will be used for encryption.

8.15.2 Returns

`Array`: The encrypted keystore v3.

8.15.3 Example

```
web3.eth.accounts.wallet.encrypt('test');
> [ { version: 3,
    id: 'dcf8ab05-a314-4e37-b972-bf9b86f91372',
    address: '06f702337909c06c82b09b7a22f0a2f0855d1f68',
    crypto:
     { ciphertext: '0de804dc63940820f6b3334e5a4bfc8214e27fb30bb7e9b7b74b25cd7eb5c604',
       cipherparams: [Object],
       cipher: 'aes-128-ctr',
       kdf: 'scrypt',
       kdfparams: [Object],
       mac: 'b2aac1485bd6ee1928665642bf8eae9ddfbc039c3a673658933d320bac6952e3' } },
  { version: 3,
    id: '9e1c7d24-b919-4428-b10e-0f3ef79f7cf0',
    address: 'b5d89661b59a9af0b34f58d19138baa2de48baaf',
    crypto:
     { ciphertext: 'd705ebed2a136d9e4db7e5ae70ed1f69d6a57370d5fbe06281eb07615f404410',
       cipherparams: [Object],
       cipher: 'aes-128-ctr',
       kdf: 'scrypt',
       kdfparams: [Object],
       mac: 'af9eca5eb01b0f70e909f824f0e7cdb90c350a802f04a9f6afe056602b92272b' } }
]
```

8.16 wallet.decrypt

```
web3.eth.accounts.wallet.decrypt(keystoreArray, password);
```

Decrypts keystore v3 objects.

8.16.1 Parameters

1. `keystoreArray` - `Array`: The encrypted keystore v3 objects to decrypt.
2. `password` - `String`: The password which will be used for encryption.

8.16.2 Returns

`Object`: The wallet object.

8.16.3 Example

```
web3.eth.accounts.wallet.decrypt([
  { version: 3,
  id: '83191a81-aaca-451f-b63d-0c5f3b849289',
  address: '06f702337909c06c82b09b7a22f0a2f0855d1f68',
  crypto:
   { ciphertext: '7d34deae112841fba86e3e6cf08f5398dda323a8e4d29332621534e2c4069e8d',
     cipherparams: { iv: '497f4d26997a84d570778eae874b2333' },
     cipher: 'aes-128-ctr',
     kdf: 'scrypt',
     kdfparams:
      { dklen: 32,
        salt: '208dd732a27aa4803bb760228dff18515d5313fd085bbce60594a3919ae2d88d',
        n: 262144,
        r: 8,
        p: 1 },
     mac: '0062a853de302513c57bfe3108ab493733034bf3cb313326f42cf26ea2619cf9' } },
   { version: 3,
  id: '7d6b91fa-3611-407b-b16b-396efb28f97e',
  address: 'b5d89661b59a9af0b34f58d19138baa2de48baaf',
  crypto:
   { ciphertext: 'cb9712d1982ff89f571fa5dbef447f14b7e5f142232bd2a913aac833730eeb43',
     cipherparams: { iv: '8cccb91cb84e435437f7282ec2ffd2db' },
     cipher: 'aes-128-ctr',
     kdf: 'scrypt',
     kdfparams:
      { dklen: 32,
        salt: '08ba6736363c5586434cd5b895e6fe41ea7db4785bd9b901dedce77a1514e8b8',
        n: 262144,
        r: 8,
        p: 1 },
     mac: 'd2eb068b37e2df55f56fa97a2bf4f55e072bef0dd703bfd917717d9dc54510f0' } }
], 'test');
> Wallet {
    0: {...},
    1: {...},
```

(continues on next page)

(continued from previous page)

```
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...},
    "0xD0122fC8DF283027b6285cc889F5aA624EaC1d23": {...}
    ...
}
```

8.17 wallet.save

```
web3.eth.accounts.wallet.save(password [, keyName]);
```

Stores the wallet encrypted and as string in local storage.

Note: Browser only.

8.17.1 Parameters

1. `password` - `String`: The password to encrypt the wallet.
2. `keyName` - `String`: (optional) The key used for the local storage position, defaults to `"web3js_wallet"`.

8.17.2 Returns

`Boolean`

8.17.3 Example

```
web3.eth.accounts.wallet.save('test#!$');
> true
```

8.18 wallet.load

```
web3.eth.accounts.wallet.load(password [, keyName]);
```

Loads a wallet from local storage and decrypts it.

Note: Browser only.

8.18.1 Parameters

1. `password` - `String`: The password to decrypt the wallet.
2. `keyName` - `String`: (optional) The key used for the localstorage position, defaults to `"web3js_wallet"`.

8.18.2 Returns

`Object`: The wallet object.

8.18.3 Example

```
web3.eth.accounts.wallet.load('test#!$', 'myWalletKey');
> Wallet {
    0: {...},
    1: {...},
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...},
    "0xD0122fC8DF283027b6285cc889F5aA624EaC1d23": {...}
    ...
}
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 9

web3.eth.personal

The `web3-eth-personal` package allows you to interact with the Ethereum node's accounts.

Note: Many of these functions send sensitive information, like password. Never call these functions over a unsecured Websocket or HTTP provider, as your password will be sent in plain text!

```
import Web3 from 'web3';
import {Personal} from 'web3-eth-personal';

// "Web3.givenProvider" will be set if in an Ethereum supported browser.
const personal = new Personal(Web3.givenProvider || 'ws://some.local-or-remote.
↪node:8546', null, options);


// or using the web3 umbrella package
const web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);

// -> web3.eth.personal
```

9.1 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

9.1.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

9.1.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

9.2 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

9.2.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block
- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

9.3 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

9.3.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

9.4 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

9.4.1 Returns

`string|number`: The current value of the defaultGasPrice property.

9.5 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

9.5.1 Returns

`string|number`: The current value of the defaultGas property.

9.6 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

9.6.1 Returns

`number`: The current value of transactionBlockTimeout

9.7 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

9.7.1 Returns

`number`: The current value of transactionConfirmationBlocks

9.8 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

9.8.1 Returns

`number`: The current value of transactionPollingTimeout

9.9 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

9.9.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

9.10 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

9.10.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

9.10.2 Returns

`Boolean`

9.10.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

9.11 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

9.11.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

9.11.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

9.12 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

9.12.1 Returns

`Object`: The given provider set or `false`.

9.12.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

9.13 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

9.13.1 Returns

`Object`: The current provider set.

9.13.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

9.14 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

9.14.1 Parameters

none

9.14.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

9.14.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

9.15 newAccount

```
web3.eth.personal.newAccount(password, [callback])
```

Create a new account on the node that Web3 is connected to with its provider. The RPC method used is `personal_newAccount`. It differs from *web3.eth.accounts.create()* where the key pair is created only on client and it's up to the developer to manage it.

Note: Never call this function over a unsecured Websocket or HTTP provider, as your password will be send in plain text!

9.15.1 Parameters

1. `password` - `String`: The password to encrypt this account with.

9.15.2 Returns

`Promise<string>` - The address of the newly created account.

9.15.3 Example

```
web3.eth.personal.newAccount('!@superpassword')
.then(console.log);
> '0x1234567891011121314151617181920212223456'
```

9.16 sign

```
web3.eth.personal.sign(dataToSign, address, password [, callback])
```

Signs data using a specific account. This data is before UTF-8 HEX decoded and enveloped as follows: `"\x19Ethereum Signed Message:\n" + message.length + message`.

Note: Sending your account password over an unsecured HTTP RPC connection is highly unsecure.

9.16.1 Parameters

1. `String` - Data to sign. If String it will be converted using *web3.utils.utf8ToHex*.
2. `String` - Address to sign data with.
3. `String` - The password of the account to sign data with.
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

9.16.2 Returns

`Promise<string>` - The signature.

9.16.3 Example

```
web3.eth.personal.sign("Hello world", "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
↪"test password!")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪"

// the below is the same
web3.eth.personal.sign(web3.utils.utf8ToHex("Hello world"),
↪"0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe", "test password!")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪"
```

9.17 ecRecover

```
web3.eth.personal.ecRecover(dataThatWasSigned, signature [, callback])
```

Recovers the account that signed the data.

9.17.1 Parameters

1. `String` - Data that was signed. If String it will be converted using *web3.utils.utf8ToHex*.
2. `String` - The signature.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

9.17.2 Returns

`Promise<string>` - The account.

9.17.3 Example

```
web3.eth.personal.ecRecover("Hello world",
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0d
↪").then(console.log);
> "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe"
```

9.18 signTransaction

```
web3.eth.personal.signTransaction(transaction, password [, callback])
```

Signs a transaction. This account needs to be unlocked.

Note: Sending your account password over an unsecured HTTP RPC connection is highly unsecure.

9.18.1 Parameters

1. `Object` - The transaction data to sign *web3.eth.sendTransaction()* for more.
2. `String` - The password of the `from` account, to sign the transaction with.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

9.18.2 Returns

`Promise<Object>` - The RLP encoded transaction. The `raw` property can be used to send the transaction using *web3.eth.sendSignedTransaction*.

9.18.3 Example

```
web3.eth.signTransaction({
    from: "0xEB014f8c8B418Db6b45774c326A0E64C78914dC0",
    gasPrice: "20000000000",
    gas: "21000",
    to: '0x3535353535353535353535353535353535353535',
    value: "1000000000000000000",
    data: ""
}, 'MyPassword!').then(console.log);
> {
    raw:
↪'0xf86c808504a817c800825208943535353535353535353535353535353535353535880de0b6b3a7640000802
↪',
    tx: {
        nonce: '0x0',
        gasPrice: '0x4a817c800',
        gas: '0x5208',
        to: '0x3535353535353535353535353535353535353535',
```

(continues on next page)

(continued from previous page)

```
        value: '0xde0b6b3a7640000',
        input: '0x',
        v: '0x25',
        r: '0x4f4c17305743700648bc4f6cd3038ec6f6af0df73e31757007b7f59df7bee88d',
        s: '0x7e1941b264348e80c78c4027afc65a87b0a5e43e86742b8ca0823584c6788fd0',
        hash: '0xda3be87732110de6c1354c83770aae630ede9ac308d9f7b399ecfba23d923384'
    }
}
```

9.19 sendTransaction

```
web3.eth.personal.sendTransaction(transactionOptions, password [, callback])
```

This method sends a transaction over the management API.

Note: Sending your account password over an unsecured HTTP RPC connection is highly unsecure.

9.19.1 Parameters

1. `Object` - The transaction options
2. `String` - The passphrase for the current account
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

9.19.2 Returns

`Promise<string>` - The transaction hash.

9.19.3 Example

```
web3.eth.sendTransaction({
    from: "0xEB014f8c8B418Db6b45774c326A0E64C78914dC0",
    gasPrice: "20000000000",
    gas: "21000",
    to: '0x3535353535353535353535353535353535353535',
    value: "1000000000000000000",
    data: ""
}, 'MyPassword!').then(console.log);
> '0xda3be87732110de6c1354c83770aae630ede9ac308d9f7b399ecfba23d923384'
```

9.20 unlockAccount

```
web3.eth.personal.unlockAccount(address, password, unlockDuraction [, callback])
```

Unlocks the given account.

Note: Sending your account password over an unsecured HTTP RPC connection is highly unsecure.

9.20.1 Parameters

1. `address` - `String`: The account address.
2. `password` - `String` - The password of the account.
3. `unlockDuration` - `Number` - The duration for the account to remain unlocked.
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

9.20.2 Returns

`Promise<boolean>` - True if the account got unlocked successful otherwise false.

9.20.3 Example

```
web3.eth.personal.unlockAccount("0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe", "test␣
↪password!", 600)
.then(console.log('Account unlocked!'));
> "Account unlocked!"
```

9.21 lockAccount

```
web3.eth.personal.lockAccount(address [, callback])
```

Locks the given account.

Note: Sending your account password over an unsecured HTTP RPC connection is highly unsecure.

9.21.1 Parameters

1. `address` - `String`: The account address. 4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

9.21.2 Returns

`Promise<boolean>`

9.21.3 Example

```
web3.eth.personal.lockAccount("0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe")
.then(console.log('Account locked!'));
> "Account locked!"
```

9.22 getAccounts

```
web3.eth.personal.getAccounts([callback])
```

Returns a list of accounts the node controls by using the provider and calling the RPC method `personal_listAccounts`. Using *web3.eth.accounts.create()* will not add accounts into this list. For that use *web3.eth.personal.newAccount()*.

The results are the same as *web3.eth.getAccounts()* except that calls the RPC method `eth_accounts`.

9.22.1 Returns

`Promise<Array>` - An array of addresses controlled by node.

9.22.2 Example

```
web3.eth.personal.getAccounts()
.then(console.log);
> ["0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
↪"0xDCc6960376d6C6dEa93647383FfB245CfCed97Cf"]
```

9.23 importRawKey

```
web3.eth.personal.importRawKey(privateKey, password)
```

Imports the given private key into the key store, encrypting it with the passphrase.

Returns the address of the new account.

Note: Sending your account password over an unsecured HTTP RPC connection is highly unsecure.

9.23.1 Parameters

1. `privateKey` - `String` - An unencrypted private key (hex string).
2. `password` - `String` - The password of the account.

9.23.2 Returns

`Promise<string>` - The address of the account.

9.23.3 Example

```
web3.eth.personal.importRawKey(
↪"cd3376bb711cb332ee3fb2ca04c6a8b9f70c316fcdf7a1f44ef4c7999483295e", "password1234")
.then(console.log);
> "0x8f337bf484b2fc75e4b0436645dcc226ee2ac531"
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 10

web3.eth.ens

The `web3.eth.ens` functions let you interacting with the Ens smart contracts.

```
import Web3 from 'web3';
import {Ens} from 'web3-eth-ens';
import {Accounts} from 'web3-eth-accounts';

// "Web3.givenProvider" will be set if in an Ethereum supported browser.
const eth = new Ens(
    Web3.givenProvider || 'ws://some.local-or-remote.node:8546',
    null,
    options
    new Accounts(Web3.givenProvider || 'ws://some.local-or-remote.node:8546', null,␣
↪options)
);


// or using the web3 umbrella package

const web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);

// -> web3.eth.ens
```

10.1 registry

```
web3.eth.ens.registry;
```

Returns the network specific Ens registry.

10.1.1 Returns

`Registry` - The current Ens registry.

10.1.2 Example

```
web3.eth.ens.registry;
> {
    ens: Ens,
    resolverContract: Contract | null,
    setProvider(provider: provider, net?: net.Socket): boolean,
    owner(name: string, callback?: (error: Error, address: string) => void): Promise
↪<string>,
    resolver(name: string): Promise<Contract>,
    checkNetwork(): Promise<string>,
}
```

10.2 resolver

```
web3.eth.ens.resolver(name);
```

Returns the resolver contract to an Ethereum address.

10.2.1 Returns

`Resolver` - The Ens resolver for this name.

10.2.2 Example

```
web3.eth.ens.resolver('ethereum.eth').then((contract) => {
    console.log(contract);
});
> Contract<Resolver>
```

10.3 supportsInterface

```
web3.eth.ens.supportsInterface(ENSName, interfaceId, [callback]);
```

Checks if the current resolver does support the desired interface.

10.3.1 Parameters

1. `ENSName` - `String`: The Ens name to resolve.
2. `interfaceId` - `String`: A defined ENS interfaceId.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

10.3.2 Returns

`Promise<boolean>` - Returns true if the given interfaceId is supported by the resolver.

10.3.3 Example

```
web3.eth.ens.supportsInterface('ethereum.eth', '0xbc1c58d1').then((supportsInterface)␣
↪=> {
    console.log(supportsInterface);
})
> true
```

10.4 getAddress

```
web3.eth.ens.getAddress(ENSName, [callback]);
```

Resolves an Ens name to an Ethereum address.

10.4.1 Parameters

1. `ENSName` - `String`: The Ens name to resolve.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

10.4.2 Returns

`Promise<string>` - The Ethereum address of the given name.

10.4.3 Example

```
web3.eth.ens.getAddress('ethereum.eth').then((address) => {
    console.log(address);
})
> 0xfB6916095ca1df60bB79Ce92cE3Ea74c37c5d359
```

10.5 setAddress

```
web3.eth.ens.setAddress(ENSName, address, options, [callback]);
```

Sets the address of an Ens name in his resolver.

10.5.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `address` - `String`: The address to set.
3. **`options` - `Object`: The options used for sending.**
    - `from` - `String`: The address the transaction should be sent from.
    - `gasPrice` - `String` (optional): The gas price in wei to use for this transaction.
    - `gas` - `Number` (optional): The maximum gas provided for this transaction (gas limit).
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

Emits an `AddrChanged` event.

10.5.2 Example

```
web3.eth.ens.setAddress(
    'ethereum.eth',
    '0xfB6916095ca1df60bB79Ce92cE3Ea74c37c5d359',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
).then((result) => {
         console.log(result.events);
});
> AddrChanged(...)

// Or using the event emitter

web3.eth.ens.setAddress(
    'ethereum.eth',
    '0xfB6916095ca1df60bB79Ce92cE3Ea74c37c5d359',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
)
.on('transactionHash', (hash) => {
    ...
})
.on('confirmation', (confirmationNumber, receipt) => {
    ...
})
.on('receipt', (receipt) => {
    ...
})
.on('error', console.error);
```

(continues on next page)

(continued from previous page)

```
// Or listen to the AddrChanged event on the resolver

web3.eth.ens.resolver('ethereum.eth').then((resolver) => {
    resolver.events.AddrChanged({fromBlock: 0}, (error, event) => {
        console.log(event);
    })
    .on('data', (event) => {
        console.log(event);
    })
    .on('changed', (event) => {
        // remove event from local database
    })
    .on('error', console.error);
});
```

For further information on the handling of contract events please see here **contract-events_**.

10.6 getPubkey

```
web3.eth.ens.getPubkey(ENSName, [callback]);
```

Returns the X and Y coordinates of the curve point for the public key.

10.6.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

10.6.2 Returns

`Object<String, String>` - The X and Y coordinates.

10.6.3 Example

```
web3.eth.ens.getPubkey('ethereum.eth').then((result) => {
    console.log(result)
});
> {
    "0": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "1": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "x": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "y": "0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

10.7 setPubkey

```
web3.eth.ens.setPubkey(ENSName, x, y, options, [callback]);
```

Sets the SECP256k1 public key associated with an Ens node

10.7.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `x` - `String`: The X coordinate of the public key.
3. `y` - `String`: The Y coordinate of the public key.
4. **`options` - `Object`: The options used for sending.**
    - `from` - `String`: The address the transaction should be sent from.
    - `gasPrice` - `String` (optional): The gas price in wei to use for this transaction.
    - `gas` - `Number` (optional): The maximum gas provided for this transaction (gas limit).
5. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

Emits an `PubkeyChanged` event.

10.7.2 Example

```
web3.eth.ens.setPubkey(
    'ethereum.eth',
    '0x0000000000000000000000000000000000000000000000000000000000000000',
    '0x0000000000000000000000000000000000000000000000000000000000000000',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
).then((result) => {
    console.log(result.events);
});
> PubkeyChanged(...)

// Or using the event emitter

web3.eth.ens.setPubkey(
    'ethereum.eth',
    '0x0000000000000000000000000000000000000000000000000000000000000000',
    '0x0000000000000000000000000000000000000000000000000000000000000000',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
)
.on('transactionHash', (hash) => {
    ...
})
.on('confirmation', (confirmationNumber, receipt) => {
    ...
})
.on('receipt', (receipt) => {
```

(continues on next page)

(continued from previous page)

```
    ...
})
.on('error', console.error);

// Or listen to the PubkeyChanged event on the resolver

web3.eth.ens.resolver('ethereum.eth').then((resolver) => {
    resolver.events.PubkeyChanged({fromBlock: 0}, function(error, event) {
        console.log(event);
    })
    .on('data', (event) => {
        console.log(event);
    })
    .on('changed', (event) => {
        // remove event from local database
    })
    .on('error', console.error);
});
```

For further information on the handling of contract events please see here **contract-events_**.

10.8 getText

```
web3.eth.ens.getText(ENSName, key, [callback]);
```

Returns the text by the given key.

10.8.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `key` - `String`: The key of the array.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

10.8.2 Returns

`Promise<string>`

10.8.3 Example

```
web3.eth.ens.getText('ethereum.eth', 'key').then((result) => {
    console.log(result);
});
> "0000000000000000000000000000000000000000000000000000000000000000"
```

10.9 setText

```
web3.eth.ens.setText(ENSName, key, value, options, [callback]);
```

Sets the content hash associated with an Ens node.

10.9.1 Parameters

1. `ENSName` - `String`: The Ens name.

2. `key` - `String`: The key. 2. `value` - `String`: The value. 3. `options` - `Object`: The options used for sending.

   - `from` - `String`: The address the transaction should be sent from.
   - `gasPrice` - `String` (optional): The gas price in wei to use for this transaction.
   - `gas` - `Number` (optional): The maximum gas provided for this transaction (gas limit).

4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

Emits an `TextChanged` event.

10.9.2 Example

```
web3.eth.ens.setText(
    'ethereum.eth',
    'key',
    'value',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
).then((result) => {
         console.log(result.events);
 });
> ContentChanged(...)

// Or using the event emitter

web3.eth.ens.setText(
    'ethereum.eth',
    'key',
    'value',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
)
.on('transactionHash', (hash) => {
    ...
})
.on('confirmation', (confirmationNumber, receipt) => {
    ...
})
.on('receipt', (receipt) => {
    ...
})
.on('error', console.error);
```

(continues on next page)

(continued from previous page)

```
// And listen to the TextChanged event on the resolver

web3.eth.ens.resolver('ethereum.eth').then((resolver) => {
    resolver.events.TextChanged({fromBlock: 0}, (error, event) => {
        console.log(event);
    })
    .on('data', (event) => {
        console.log(event);
    })
    .on('changed', (event) => {
        // remove event from local database
    })
    .on('error', console.error);
});
```

For further information on the handling of contract events please see here **contract-events_**.

10.10 getContent

```
web3.eth.ens.getContent(ENSName, [callback]);
```

Returns the content hash associated with an Ens node.

10.10.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

10.10.2 Returns

`Promise<string>` - The content hash associated with an Ens node.

10.10.3 Example

```
web3.eth.ens.getContent('ethereum.eth').then((result) => {
    console.log(result);
});
> "0x0000000000000000000000000000000000000000000000000000000000000000"
```

10.11 setContent

```
web3.eth.ens.setContent(ENSName, hash, options, [callback]);
```

Sets the content hash associated with an Ens node.

10.11.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `hash` - `String`: The content hash to set.
3. **`options` - `Object`: The options used for sending.**
    - `from` - `String`: The address the transaction should be sent from.
    - `gasPrice` - `String` (optional): The gas price in wei to use for this transaction.
    - `gas` - `Number` (optional): The maximum gas provided for this transaction (gas limit).
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

Emits an `ContentChanged` event.

10.11.2 Example

```
web3.eth.ens.setContent(
    'ethereum.eth',
    '0x0000000000000000000000000000000000000000000000000000000000000000',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
).then((result) => {
         console.log(result.events);
 });
> ContentChanged(...)

// Or using the event emitter

web3.eth.ens.setContent(
    'ethereum.eth',
    '0x0000000000000000000000000000000000000000000000000000000000000000',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
)
.on('transactionHash', (hash) => {
    ...
})
.on('confirmation', (confirmationNumber, receipt) => {
    ...
})
.on('receipt', (receipt) => {
    ...
})
.on('error', console.error);

// Or listen to the ContentChanged event on the resolver
```

(continues on next page)

(continued from previous page)

```
web3.eth.ens.resolver('ethereum.eth').then((resolver) => {
    resolver.events.ContentChanged({fromBlock: 0}, (error, event) => {
        console.log(event);
    })
    .on('data', (event) => {
        console.log(event);
    })
    .on('changed', (event) => {
        // remove event from local database
    })
    .on('error', console.error);
});
```

For further information on the handling of contract events please see here **contract-events_**.

10.12 getMultihash

```
web3.eth.ens.getMultihash(ENSName, [callback]);
```

Returns the multihash associated with an Ens node.

10.12.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

10.12.2 Returns

`Promise<string>` - The associated multihash.

10.12.3 Example

```
web3.eth.ens.getMultihash('ethereum.eth').then((result) => {
    console.log(result);
});
> 'QmXpSwxdmgWaYrgMUzuDWCnjsZo5RxphE3oW7VhTMSCoKK'
```

10.13 setMultihash

```
web3.eth.ens.setMultihash(ENSName, hash, options, [callback]);
```

Sets the multihash associated with an Ens node.

10.13.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `hash` - `String`: The multihash to set.
3. **`options` - `Object`: The options used for sending.**
    - `from` - `String`: The address the transaction should be sent from.
    - `gasPrice` - `String` (optional): The gas price in wei to use for this transaction.
    - `gas` - `Number` (optional): The maximum gas provided for this transaction (gas limit).
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

Emits an ‘‘MultihashChanged‘‘event.

10.13.2 Example

```
web3.eth.ens.setMultihash(
    'ethereum.eth',
    'QmXpSwxdmgWaYrgMUzuDWCnjsZo5RxphE3oW7VhTMSCoKK',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
).then((result) => {
    console.log(result.events);
});
> MultihashChanged(...)

// Or using the event emitter

web3.eth.ens.setMultihash(
    'ethereum.eth',
    'QmXpSwxdmgWaYrgMUzuDWCnjsZo5RxphE3oW7VhTMSCoKK',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
)
.on('transactionHash', (hash) => {
    ...
})
.on('confirmation', (confirmationNumber, receipt) => {
    ...
})
.on('receipt', (receipt) => {
    ...
})
.on('error', console.error);
```

For further information on the handling of contract events please see here **contract-events_**.

10.14 getContenthash

```
web3.eth.ens.getContenthash(ENSName, [callback]);
```

Returns the contenthash associated with an Ens node. *contenthash* encoding is defined in [EIP1577](http://eips.ethereum.org/EIPS/eip-1577)

10.14.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

10.14.2 Returns

`Promise<string>` - The associated contenthash.

10.14.3 Example

```
web3.eth.ens.getContenthash('pac-txt.eth').then((result) => {
    console.log(result);
});
> '0xe30101701220e08ea2458249e8f26aee72b95b39c33849a992a3eff40bd06d26c12197adef16'
```

10.15 setContenthash

```
web3.eth.ens.setContenthash(ENSName, hash, options, [callback]);
```

Sets the contenthash associated with an Ens node.

10.15.1 Parameters

1. `ENSName` - `String`: The Ens name.
2. `hash` - `String`: The contenthash to set.
3. **`options` - `Object`: The options used for sending.**
    - `from` - `String`: The address the transaction should be sent from.
    - `gasPrice` - `String` (optional): The gas price in wei to use for this transaction.
    - `gas` - `Number` (optional): The maximum gas provided for this transaction (gas limit).
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

Emits an `ContenthashChanged` event.

10.15.2 Example

```
web3.eth.ens.setContenthash(
    'ethereum.eth',
    '0xe301017012208cd82588c4e08268fa0b824caa93847ac843410076eeedc41d65fb52eccbb9e6',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
).then((result) => {
    console.log(result.events);
});
> ContenthashChanged(...)

// Or using the event emitter

web3.eth.ens.setContenthash(
    'ethereum.eth',
    '0xe301017012208cd82588c4e08268fa0b824caa93847ac843410076eeedc41d65fb52eccbb9e6',
    {
        from: '0x9CC9a2c777605Af16872E0997b3Aeb91d96D5D8c'
    }
)
.on('transactionHash', (hash) => {
    ...
})
.on('confirmation', (confirmationNumber, receipt) => {
    ...
})
.on('receipt', (receipt) => {
    ...
})
.on('error', console.error);
```

For further information on the handling of contract events please see here **contract-events_**.

10.16 Ens events

The Ens API provides the possibility for listening to all Ens related events.

10.16.1 Known resolver events

1. `AddrChanged` - AddrChanged(node bytes32, a address)
2. `ContentChanged` - ContentChanged(node bytes32, hash bytes32)
3. `NameChanged` - NameChanged(node bytes32, name string)
4. `ABIChanged` - ABIChanged(node bytes32, contentType uint256)
5. `PubkeyChanged` - PubkeyChanged(node bytes32, x bytes32, y bytes32)
6. `TextChanged` - TextChanged(bytes32 indexed node, string indexedKey, string key)
7. `ContenthashChanged` - ContenthashChanged(bytes32 indexed node, bytes hash)

10.16.2 Example

```
web3.eth.ens.resolver('ethereum.eth').then((resolver) => {
    resolver.events.AddrChanged({fromBlock: 0}, (error, event) => {
        console.log(event);
    })
    .on('data', (event) => {
        console.log(event);
    })
    .on('changed', (event) => {
        // remove event from local database
    })
    .on('error', console.error);
});
> {
    returnValues: {
        node: '0x123456789...',
        a: '0x123456789...',
    },
    raw: {
        data: '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
        topics: [
            '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
            '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385'
        ]
    },
    event: 'AddrChanged',
    signature: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    logIndex: 0,
    transactionIndex: 0,
    transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    blockHash: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    blockNumber: 1234,
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
}
```

10.16.3 Known registry events

1. `Transfer` - Transfer(node bytes32, owner address)

2. `NewOwner` - NewOwner(node bytes32, label bytes32, owner address) 4. `NewResolver` - NewResolver(node bytes32, resolver address) 5. `NewTTL` - NewTTL(node bytes32, ttl uint64)

10.16.4 Example

```
web3.eth.ens.resistry.then((registry) => {
    registry.events.Transfer({fromBlock: 0}, (error, event) => {
          console.log(event);
      })
      .on('data', (event) => {
          console.log(event);
      })
      .on('changed', (event) => {
```

(continues on next page)

(continued from previous page)

```
        // remove event from local database
    })
    .on('error', console.error);
});
> {
    returnValues: {
        node: '0x123456789...',
        owner: '0x123456789...',
    },
    raw: {
        data: '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
        topics: [
            '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
            '0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385'
        ]
    },
    event: 'Transfer',
    signature: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    logIndex: 0,
    transactionIndex: 0,
    transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    blockHash: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    blockNumber: 1234,
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
}
```

For further information on the handling of contract events please see here **contract-events_**.

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 11

web3.eth.Iban

The `web3.eth.Iban` function lets convert Ethereum addresses from and to IBAN and BBAN.

```
import {Iban} from 'web3-eth-iban';

const iban = new Iban('XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS');

// or using the web3 umbrella package

import Web3 from 'web3';
const web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);

// -> new web3.eth.Iban('XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS')
```

11.1 Iban instance

This's instance of Iban

```
> Iban { _iban: 'XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS' }
```

11.2 toAddress

static function

```
web3.eth.Iban.toAddress(ibanAddress)
```

Singleton: Converts a direct IBAN address into an Ethereum address.

Note: This method also exists on the IBAN instance.

11.2.1 Parameters

1. `String`: the IBAN address to convert.

11.2.2 Returns

`String` - The Ethereum address.

11.2.3 Example

```
web3.eth.Iban.toAddress("XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS");
> "0x00c5496aEe77C1bA1f0854206A26DdA82a81D6D8"
```

11.3 toIban

static function

```
web3.eth.Iban.toIban(address)
```

Singleton: Converts an Ethereum address to a direct IBAN address.

11.3.1 Parameters

1. `String`: the Ethereum address to convert.

11.3.2 Returns

`String` - The IBAN address.

11.3.3 Example

```
web3.eth.Iban.toIban("0x00c5496aEe77C1bA1f0854206A26DdA82a81D6D8");
> "XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS"
```

static function, return IBAN instance

11.4 fromAddress

```
web3.eth.Iban.fromAddress(address)
```

Singleton: Converts an Ethereum address to a direct IBAN instance.

11.4.1 Parameters

1. `String`: the Ethereum address to convert.

11.4.2 Returns

`Object` - The IBAN instance.

11.4.3 Example

```
web3.eth.Iban.fromAddress("0x00c5496aEe77C1bA1f0854206A26DdA82a81D6D8");
> Iban {_iban: "XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS"}
```

---

static function, return IBAN instance

11.5 fromBban

```
web3.eth.Iban.fromBban(bbanAddress)
```

Singleton: Converts an BBAN address to a direct IBAN instance.

11.5.1 Parameters

1. `String`: the BBAN address to convert.

11.5.2 Returns

`Object` - The IBAN instance.

11.5.3 Example

```
web3.eth.Iban.fromBban('ETHXREGGAVOFYORK');
> Iban {_iban: "XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS"}
```

---

static function, return IBAN instance

11.6 createIndirect

```
web3.eth.Iban.createIndirect(options)
```

Singleton: Creates an indirect IBAN address from a institution and identifier.

11.6.1 Parameters

1. **`Object`: the options object as follows:**
    - `institution` - `String`: the institution to be assigned
    - `identifier` - `String`: the identifier to be assigned

11.6.2 Returns

`Object` - The IBAN instance.

11.6.3 Example

```
web3.eth.Iban.createIndirect({
    institution: "XREG",
    identifier: "GAVOFYORK"
});
> Iban {_iban: "XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS"}
```

---

static function, return boolean

11.7 isValid

```
web3.eth.Iban.isValid(ibanAddress)
```

Singleton: Checks if an IBAN address is valid.

Note: This method also exists on the IBAN instance.

11.7.1 Parameters

1. `String`: the IBAN address to check.

11.7.2 Returns

`Boolean`

11.7.3 Example

```
web3.eth.Iban.isValid("XE81ETHXREGGAVOFYORK");
> true

web3.eth.Iban.isValid("XE82ETHXREGGAVOFYORK");
> false // because the checksum is incorrect
```

11.8 prototype.isValid

method of Iban instance

```
web3.eth.Iban.prototype.isValid()
```

Singleton: Checks if an IBAN address is valid.

Note: This method also exists on the IBAN instance.

11.8.1 Parameters

1. `String`: the IBAN address to check.

11.8.2 Returns

`Boolean`

11.8.3 Example

```
const iban = new web3.eth.Iban("XE81ETHXREGGAVOFYORK");
iban.isValid();
> true
```

11.9 prototype.isDirect

method of Iban instance

```
web3.eth.Iban.prototype.isDirect()
```

Checks if the IBAN instance is direct.

11.9.1 Returns

`Boolean`

11.9.2 Example

```
const iban = new web3.eth.Iban("XE81ETHXREGGAVOFYORK");
iban.isDirect();
> false
```

11.10 prototype.isIndirect

method of Iban instance

```
web3.eth.Iban.prototype.isIndirect()
```

Checks if the IBAN instance is indirect.

11.10.1 Returns

`Boolean`

11.10.2 Example

```
const iban = new web3.eth.Iban("XE81ETHXREGGAVOFYORK");
iban.isIndirect();
> true
```

11.11 prototype.checksum

method of Iban instance

```
web3.eth.Iban.prototype.checksum()
```

Returns the checksum of the IBAN instance.

11.11.1 Returns

`String`: The checksum of the IBAN

11.11.2 Example

```
const iban = new web3.eth.Iban("XE81ETHXREGGAVOFYORK");
iban.checksum();
> "81"
```

11.12 prototype.institution

method of Iban instance

```
web3.eth.Iban.prototype.institution()
```

Returns the institution of the IBAN instance.

11.12.1 Returns

`String`: The institution of the IBAN

11.12.2 Example

```
const iban = new web3.eth.Iban("XE81ETHXREGGAVOFYORK");
iban.institution();
> 'XREG'
```

11.13 prototype.client

method of Iban instance

```
web3.eth.Iban.prototype.client()
```

Returns the client of the IBAN instance.

11.13.1 Returns

`String`: The client of the IBAN

11.13.2 Example

```
const iban = new web3.eth.Iban("XE81ETHXREGGAVOFYORK");
iban.client();
> 'GAVOFYORK'
```

11.14 prototype.toAddress

method of Iban instance

```
web3.eth.Iban.prototype.toString()
```

Returns the Ethereum address of the IBAN instance.

11.14.1 Returns

`String`: The Ethereum address of the IBAN

11.14.2 Example

```
const iban = new web3.eth.Iban('XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS');
iban.toAddress();
> '0x00c5496aEe77C1bA1f0854206A26DdA82a81D6D8'
```

11.15 prototype.toString

method of Iban instance

```
web3.eth.Iban.prototype.toString()
```

Returns the IBAN address of the IBAN instance.

11.15.1 Returns

`String`: The IBAN address.

11.15.2 Example

```
const iban = new web3.eth.Iban('XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS');
iban.toString();
> 'XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS'
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 12

web3.eth.net

Functions to receive details about the current connected network.

12.1 getId

```
web3.eth.net.getId([callback])
web3.shh.net.getId([callback])
```

Gets the current network ID.

12.1.1 Parameters

none

12.1.2 Returns

`Promise` returns `Number`: The network ID.

12.1.3 Example

```
web3.eth.net.getId().then(console.log);
> 1
```

12.2 isListening

```
web3.eth.net.isListening([callback])
web3.shh.net.isListening([callback])
```

Checks if the node is listening for peers.

12.2.1 Parameters

none

12.2.2 Returns

`Promise` returns `Boolean`

12.2.3 Example

```
web3.eth.isListening().then(console.log);
> true
```

12.3 getPeerCount

```
web3.eth.net.getPeerCount([callback])
web3.shh.net.getPeerCount([callback])
```

Get the number of peers connected to.

12.3.1 Parameters

none

12.3.2 Returns

`Promise` returns `Number`

12.3.3 Example

```
web3.eth.getPeerCount().then(console.log);
> 25
```

12.4 getNetworkType

```
web3.eth.net.getNetworkType([callback])
```

Guesses the chain the node is connected by comparing the genesis hashes.

Note: It's recommended to use the *web3.eth.getChainId* method to detect the currently connected chain.

12.4.1 Returns

`Promise` returns `String`:

- `"main"` for main network
- `"morden"` for the morden test network
- `"rinkeby"` for the rinkeby test network
- `"ropsten"` for the ropsten test network
- `"kovan"` for the kovan test network
- `"private"` for undetectable networks.

12.4.2 Example

```
web3.eth.net.getNetworkType().then(console.log);
> "main"
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 13

web3.eth.abi

The `web3-eth-abi` package allows you to de- and encode parameters from a ABI (Application Binary Interface). This will be used for calling functions of a deployed smart-contract.

```
import {AbiCoder} from 'web3-eth-abi';

const abiCoder = new AbiCoder();


// or using the web3 umbrella package


import Web3 from 'web3';

const web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);
// -> web3.eth.abi
```

13.1 encodeFunctionSignature

```
web3.eth.abi.encodeFunctionSignature(functionName);
```

Encodes the function name to its ABI signature, which are the first 4 bytes of the sha3 hash of the function name including types.

13.1.1 Parameters

1. `functionName` - `String|Object`: The function name to encode. or the JSON interface object of the function. If string it has to be in the form `function(type,type,...)`, e.g: `myFunction(uint256,uint32[],bytes10,bytes)`

13.1.2 Returns

`String` - The ABI signature of the function.

13.1.3 Example

```
// From a JSON interface object
web3.eth.abi.encodeFunctionSignature({
    name: 'myMethod',
    type: 'function',
    inputs: [{
        type: 'uint256',
        name: 'myNumber'
    },{
        type: 'string',
        name: 'myString'
    }]
})
> 0x24ee0097

// Or string
web3.eth.abi.encodeFunctionSignature('myMethod(uint256,string)')
> '0x24ee0097'
```

13.2 encodeEventSignature

```
web3.eth.abi.encodeEventSignature(eventName);
```

Encodes the event name to its ABI signature, which are the sha3 hash of the event name including input types.

13.2.1 Parameters

1. `eventName` - `String|Object`: The event name to encode. or the JSON interface object of the event. If string it has to be in the form `event(type,type,...)`, e.g: `myEvent(uint256,uint32[],bytes10,bytes)`

13.2.2 Returns

`String` - The ABI signature of the event.

13.2.3 Example

```
web3.eth.abi.encodeEventSignature('myEvent(uint256,bytes32)')
> 0xf2eeb729e636a8cb783be044acf6b7b1e2c5863735b60d6daae84c366ee87d97

// or from a json interface object
web3.eth.abi.encodeEventSignature({
    name: 'myEvent',
```

(continues on next page)

(continued from previous page)

```
    type: 'event',
    inputs: [{
        type: 'uint256',
        name: 'myNumber'
    },{
        type: 'bytes32',
        name: 'myBytes'
    }]
})
> 0xf2eeb729e636a8cb783be044acf6b7b1e2c5863735b60d6daae84c366ee87d97
```

13.3 encodeParameter

```
web3.eth.abi.encodeParameter(type, parameter);
```

Encodes a parameter based on its type to its ABI representation.

13.3.1 Parameters

1. `type` - `String|Object`: The type of the parameter, see the solidity documentation for a list of types.
2. `parameter` - `Mixed`: The actual parameter to encode.

13.3.2 Returns

`String` - The ABI encoded parameter.

13.3.3 Example

```
web3.eth.abi.encodeParameter('uint256', '2345675643');
> "0x000000000000000000000000000000000000000000000000000000008bd02b7b"

web3.eth.abi.encodeParameter('uint256', '2345675643');
> "0x000000000000000000000000000000000000000000000000000000008bd02b7b"

web3.eth.abi.encodeParameter('bytes32', '0xdf3234');
> "0xdf3234000000000000000000000000000000000000000000000000000000000000"

web3.eth.abi.encodeParameter('bytes', '0xdf3234');
>
↪"0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪"

web3.eth.abi.encodeParameter('bytes32[]', ['0xdf3234', '0xfdfd']);
>
↪"0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪"
```

13.4 encodeParameters

```
web3.eth.abi.encodeParameters(typesArray, parameters);
```

Encodes a function parameters based on its JSON interface object.

13.4.1 Parameters

1. `typesArray` - `Array<String|Object>|Object`: An array with types or a JSON interface of a function. See the solidity documentation for a list of types.
2. `parameters` - `Array`: The parameters to encode.

13.4.2 Returns

`String` - The ABI encoded parameters.

13.4.3 Example

```
web3.eth.abi.encodeParameters(['uint256','string'], ['2345675643', 'Hello!%']);
>
↪"0x0000000000000000000000000000000000000000000000000000000008bd02b7b000000000000000000000000000000000000
↪"

web3.eth.abi.encodeParameters(['uint8[]','bytes32'], [['34','434'], '0x324567fff']);
> "0x0"
```

13.5 encodeFunctionCall

```
web3.eth.abi.encodeFunctionCall(jsonInterface, parameters);
```

Encodes a function call using its JSON interface object and given parameters.

13.5.1 Parameters

1. `jsonInterface` - `Object`: The JSON interface object of a function.
2. `parameters` - `Array`: The parameters to encode.

13.5.2 Returns

`String` - The ABI encoded function call. Means function signature + parameters.

13.5.3 Example

```
web3.eth.abi.encodeFunctionCall({
    name: 'myMethod',
    type: 'function',
    inputs: [{
        type: 'uint256',
        name: 'myNumber'
    },{
        type: 'string',
        name: 'myString'
    }]
}, ['2345675643', 'Hello!%']);
>
↪"0x24ee0097000000000000000000000000000000000000000000000000000000008bd02b7b0000000000000000000000000
↪"
```

13.6 decodeParameter

```
web3.eth.abi.decodeParameter(type, hexString);
```

Decodes an ABI encoded parameter to its JavaScript type.

13.6.1 Parameters

1. `type` - `String|Object`: The type of the parameter, see the solidity documentation for a list of types.
2. `hexString` - `String`: The ABI byte code to decode.

13.6.2 Returns

`Mixed` - The decoded parameter.

13.6.3 Example

```
web3.eth.abi.decodeParameter('uint256',
↪'0x0000000000000000000000000000000000000000000000000000000000000010');
> "16"

web3.eth.abi.decodeParameter('string',
↪'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪');
> "Hello!%!"

web3.eth.abi.decodeParameter('string',
↪'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪');
> "Hello!%!"
```

13.7 decodeParameters

```
web3.eth.abi.decodeParameters(typesArray, hexString);
```

Decodes ABI encoded parameters to its JavaScript types.

13.7.1 Parameters

1. `typesArray` - `Array<String|Object>|Object`: An array with types or a JSON interface outputs array. See the solidity documentation for a list of types.
2. `hexString` - `String`: The ABI byte code to decode.

13.7.2 Returns

`Object` - The result object containing the decoded parameters.

13.7.3 Example

```
web3.eth.abi.decodeParameters(['string', 'uint256'],
↪'0x0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000
↪');
> Result { '0': 'Hello!%!', '1': '234' }

web3.eth.abi.decodeParameters([{
    type: 'string',
    name: 'myString'
},{
    type: 'uint256',
    name: 'myNumber'
}],
↪'0x0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000
↪');
> Result {
    '0': 'Hello!%!',
    '1': '234',
    myString: 'Hello!%!',
    myNumber: '234'
}
```

13.8 decodeLog

```
web3.eth.abi.decodeLog(inputs, hexString, topics);
```

Decodes ABI encoded log data and indexed topic data.

13.8.1 Parameters

1. `inputs` - `Array`: A JSON interface inputs array. See the solidity documentation for a list of types.
2. `hexString` - `String`: The ABI byte code in the `data` field of a log.
3. `topics` - `Array`: An array with the index parameter topics of the log, without the topic[0] if its a non-anonymous event, otherwise with topic[0].

13.8.2 Returns

`Object` - The result object containing the decoded parameters.

13.8.3 Example

```
web3.eth.abi.decodeLog([{
    type: 'string',
    name: 'myString'
},{
    type: 'uint256',
    name: 'myNumber',
    indexed: true
},{
    type: 'uint8',
    name: 'mySmallNumber',
    indexed: true
}],

↪'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪',
['0x000000000000000000000000000000000000000000000000000000000000f310',
↪'0x0000000000000000000000000000000000000000000000000000000000000010']);
> Result {
    '0': 'Hello%!',
    '1': '62224',
    '2': '16',
    myString: 'Hello%!',
    myNumber: '62224',
    mySmallNumber: '16'
}
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 14

web3.*.net

The `web3-net` package allows you to interact with the Ethereum nodes network properties.

```
import Web3 from 'web3';
import {Net} from 'web3-net';

// "Personal.providers.givenProvider" will be set if in an Ethereum supported browser.
const net = new Net(Web3.givenProvider || 'ws://some.local-or-remote.node:8546', null,
↪ options);


// or using the web3 umbrella package
const web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);

// -> web3.eth.net
// -> web3.shh.net
```

14.1 getId

```
web3.eth.net.getId([callback])
web3.shh.net.getId([callback])
```

Gets the current network ID.

14.1.1 Parameters

none

14.1.2 Returns

`Promise` returns `Number`: The network ID.

14.1.3 Example

```
web3.eth.net.getId().then(console.log);
> 1
```

14.2 isListening

```
web3.eth.net.isListening([callback])
web3.shh.net.isListening([callback])
```

Checks if the node is listening for peers.

14.2.1 Parameters

none

14.2.2 Returns

`Promise` returns `Boolean`

14.2.3 Example

```
web3.eth.isListening().then(console.log);
> true
```

14.3 getPeerCount

```
web3.eth.net.getPeerCount([callback])
web3.shh.net.getPeerCount([callback])
```

Get the number of peers connected to.

14.3.1 Parameters

none

14.3.2 Returns

`Promise` returns `Number`

14.3.3 Example

```
web3.eth.getPeerCount().then(console.log);
> 25
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 15

web3.bzz

The `web3-bzz` does no longer exists in the web3.js project. Check out the Swarm Docs for seeing possible alternatives to interact with the Swarm API.

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 16

web3.shh

The `web3-shh` package allows you to interact with the whisper protocol for broadcasting. For more see Whisper Overview.

```
import Web3 from 'web3';
import {Shh} import 'web3-shh';

// "Web3.givenProvider" will be set if in an Ethereum supported browser.
const shh = new Shh(Web3.givenProvider || 'ws://some.local-or-remote.node:8546', null,
↪ options);


// or using the web3 umbrella package
const web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options;

// -> web3.shh
```

16.1 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

16.1.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

16.1.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

16.2 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

16.2.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block

- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

16.3 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

16.3.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

16.4 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

16.4.1 Returns

`string|number`: The current value of the defaultGasPrice property.

16.5 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

16.5.1 Returns

`string|number`: The current value of the defaultGas property.

16.6 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

16.6.1 Returns

`number`: The current value of transactionBlockTimeout

16.7 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

16.7.1 Returns

`number`: The current value of transactionConfirmationBlocks

16.8 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

16.8.1 Returns

`number`: The current value of transactionPollingTimeout

16.9 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

16.9.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

16.10 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

16.10.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

16.10.2 Returns

`Boolean`

16.10.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

16.11 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

16.11.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

16.11.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

16.12 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

16.12.1 Returns

`Object`: The given provider set or `false`.

16.12.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

16.13 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

16.13.1 Returns

`Object`: The current provider set.

16.13.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

16.14 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

16.14.1 Parameters

none

16.14.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

16.14.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

16.15 getId

```
web3.eth.net.getId([callback])
web3.shh.net.getId([callback])
```

Gets the current network ID.

16.15.1 Parameters

none

16.15.2 Returns

`Promise` returns `Number`: The network ID.

16.15.3 Example

```
web3.eth.net.getId().then(console.log);
> 1
```

16.16 isListening

```
web3.eth.net.isListening([callback])
web3.shh.net.isListening([callback])
```

Checks if the node is listening for peers.

16.16.1 Parameters

none

16.16.2 Returns

`Promise` returns `Boolean`

16.16.3 Example

```
web3.eth.isListening().then(console.log);
> true
```

16.17 getPeerCount

```
web3.eth.net.getPeerCount([callback])
web3.shh.net.getPeerCount([callback])
```

Get the number of peers connected to.

16.17.1 Parameters

none

16.17.2 Returns

`Promise` returns `Number`

16.17.3 Example

```
web3.eth.getPeerCount().then(console.log);
> 25
```

16.18 getVersion

```
web3.shh.getVersion([callback])
```

Returns the version of the running whisper.

16.18.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.18.2 Returns

`Promise<string>` - The version of the current whisper running.

16.18.3 Example

```
web3.shh.getVersion()
.then(console.log);
> "5.0"
```

16.19 getInfo

```
web3.shh.getInfo([callback])
```

Gets information about the current whisper node.

16.19.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.19.2 Returns

`Promise<Object>` - The information of the node with the following properties:

- `messages` - `Number`: Number of currently floating messages.
- `maxMessageSize` - `Number`: The current message size limit in bytes.
- `memory` - `Number`: The memory size of the floating messages in bytes.
- `minPow` - `Number`: The current minimum PoW requirement.

16.19.3 Example

```
web3.shh.getInfo().then(console.log);
> {
    "minPow": 0.8,
    "maxMessageSize": 12345,
    "memory": 1234335,
    "messages": 20
}
```

16.20 setMaxMessageSize

```
web3.shh.setMaxMessageSize(size, [callback])
```

Sets the maximal message size allowed by this node. Incoming and outgoing messages with a larger size will be rejected. Whisper message size can never exceed the limit imposed by the underlying P2P protocol (10 Mb).

16.20.1 Parameters

1. `Number` - Message size in bytes.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.20.2 Returns

`Promise<boolean>` - Returns `true` on success, error on failure.

16.20.3 Example

```
web3.shh.setMaxMessageSize(1234565)
.then(console.log);
> true
```

16.21 setMinPoW

```
web3.shh.setMinPoW(pow, [callback])
```

Sets the minimal PoW required by this node.

This experimental function was introduced for the future dynamic adjustment of PoW requirement. If the node is overwhelmed with messages, it should raise the PoW requirement and notify the peers. The new value should be set relative to the old value (e.g. double). The old value can be obtained via *web3.shh.getInfo()*.

16.21.1 Parameters

1. `Number` - The new PoW requirement.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.21.2 Returns

`Promise<boolean>` - Returns `true` on success, error on failure.

16.21.3 Example

```
web3.shh.setMinPoW(0.9)
.then(console.log);
> true
```

16.22 markTrustedPeer

```
web3.shh.markTrustedPeer(enode, [callback])
```

Marks specific peer trusted, which will allow it to send historic (expired) messages.

Note: This function is not adding new nodes, the node needs to be an existing peer.

16.22.1 Parameters

1. `String` - Enode of the trusted peer.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.22.2 Returns

`Promise<boolean>` - Returns `true` on success, error on failure.

16.22.3 Example

```
web3.shh.markTrustedPeer()
.then(console.log);
> true
```

16.23 newKeyPair

```
web3.shh.newKeyPair([callback])
```

Generates a new public and private key pair for message decryption and encryption.

16.23.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.23.2 Returns

`Promise<string>` - Returns the Key ID on success and an error on failure.

16.23.3 Example

```
web3.shh.newKeyPair()
.then(console.log);
> "5e57b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f"
```

16.24 addPrivateKey

```
web3.shh.addPrivateKey(privateKey, [callback])
```

Stores a key pair derived from a private key, and returns its ID.

16.24.1 Parameters

1. `String` - The private key as HEX bytes to import.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.24.2 Returns

`Promise<string>` - The Key ID on success and an error on failure.

16.24.3 Example

```
web3.shh.addPrivateKey(
↪'0x8bda3abeb454847b515fa9b404cede50b1cc63cfdeddd4999d074284b4c21e15')
.then(console.log);
> "3e22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f"
```

16.25 deleteKeyPair

```
web3.shh.deleteKeyPair(id, [callback])
```

Deletes the specifies key if it exists.

16.25.1 Parameters

1. `String` - The key pair ID, returned by the creation functions (`shh.newKeyPair` and `shh.addPrivateKey`).
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.25.2 Returns

`Promise<boolean>` - Returns `true` on success, error on failure.

16.25.3 Example

```
web3.shh.deleteKeyPair(
↪'3e22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
> true
```

16.26 hasKeyPair

```
web3.shh.hasKeyPair(id, [callback])
```

Checks if the whisper node has a private key of a key pair matching the given ID.

16.26.1 Parameters

1. `String` - The key pair ID, returned by the creation functions (`shh.newKeyPair` and `shh.addPrivateKey`).
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.26.2 Returns

`Promise<boolean>` - Returns `true` on if the key pair exist in the node, `false` if not. Error on failure.

16.26.3 Example

```
web3.shh.hasKeyPair('fe22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f
↪')
.then(console.log);
> true
```

16.27 getPublicKey

```
web3.shh.getPublicKey(id, [callback])
```

Returns the public key for a key pair ID.

16.27.1 Parameters

1. `String` - The key pair ID, returned by the creation functions (`shh.newKeyPair` and `shh.addPrivateKey`).
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.27.2 Returns

`Promise<string>` - Returns the Public key on success and an error on failure.

16.27.3 Example

```
web3.shh.getPublicKey(
↪'3e22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
>
↪"0x04d1574d4eab8f3dde4d2dc7ed2c4d699d77cbbdd09167b8fffa099652ce4df00c4c6e0263eafe05007a46fdf0c8d32
↪"
```

16.28 getPrivateKey

```
web3.shh.getPrivateKey(id, [callback])
```

Returns the private key for a key pair ID.

16.28.1 Parameters

1. `String` - The key pair ID, returned by the creation functions (`shh.newKeyPair` and `shh.addPrivateKey`).
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.28.2 Returns

`Promise<string>` - Returns the private key on success and an error on failure.

16.28.3 Example

```
web3.shh.getPrivateKey(
↪'3e22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
> "0x234234e22b9ffc2387e18636e0534534a3d0c56b024356743245326 4c16e78a2adc"
```

16.29 newSymKey

```
web3.shh.newSymKey([callback])
```

Generates a random symmetric key and stores it under an ID, which is then returned. Will be used for encrypting and decrypting of messages where the sym key is known to both parties.

16.29.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.29.2 Returns

`Promise<string>` - Returns the Key ID on success and an error on failure.

16.29.3 Example

```
web3.shh.newSymKey()
.then(console.log);
> "cec94d139ff51d7df1d228812b90c23ec1f909afa0840ed80f1e04030bb681e4"
```

16.30 addSymKey

```
web3.shh.addSymKey(symKey, [callback])
```

Stores the key, and returns its ID.

16.30.1 Parameters

1. `String` - The raw key for symmetric encryption as HEX bytes.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.30.2 Returns

`Promise<string>` - Returns the key ID on success and an error on failure.

16.30.3 Example

```
web3.shh.addSymKey('0x5e11b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f
↪')
.then(console.log);
> "fea94d139ff51d7df1d228812b90c23ec1f909afa0840ed80f1e04030bb681e4"
```

16.31 generateSymKeyFromPassword

```
web3.shh.generateSymKeyFromPassword(password, [callback])
```

Generates the key from password, stores it, and returns its ID.

16.31.1 Parameters

1. `String` - A password to generate the sym key from.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.31.2 Returns

`Promise<String|Error>` - Returns the Key ID on success and an error on failure.

16.31.3 Example

```
web3.shh.generateSymKeyFromPassword('Never use this password - password!')
.then(console.log);
> "2e57b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f"
```

16.32 hasSymKey

```
web3.shh.hasSymKey(id, [callback])
```

Checks if there is a symmetric key stored with the given ID.

16.32.1 Parameters

1. `String` - The key pair ID, returned by the creation functions (`shh.newSymKey`, `shh.addSymKey` or `shh.generateSymKeyFromPassword`).
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.32.2 Returns

`Promise<boolean>` - Returns `true` on if the symmetric key exist in the node, `false` if not. Error on failure.

16.32.3 Example

```
web3.shh.hasSymKey('f6dcf21ed6a17bd78d8c4c63195ab997b3b65ea683705501eae82d32667adc92')
.then(console.log);
> true
```

16.33 getSymKey

```
web3.shh.getSymKey(id, [callback])
```

Returns the symmetric key associated with the given ID.

16.33.1 Parameters

1. `String` - The key pair ID, returned by the creation functions (`shh.newKeyPair` and `shh.addPrivateKey`).
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.33.2 Returns

`Promise<string>` - Returns the raw symmetric key on success and an error on failure.

16.33.3 Example

```
web3.shh.getSymKey('af33b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
> "0xa82a520aff70f7a989098376e48ec128f25f767085e84d7fb995a9815eebff0a"
```

16.34 deleteSymKey

```
web3.shh.deleteSymKey(id, [callback])
```

Deletes the symmetric key associated with the given ID.

16.34.1 Parameters

1. `String` - The key pair ID, returned by the creation functions (`shh.newKeyPair` and `shh.addPrivateKey`).
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

16.34.2 Returns

`Promise<boolean>` - Returns `true` on if the symmetric key was deleted, error on failure.

16.34.3 Example

```
web3.shh.deleteSymKey(
↪'bf31b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
> true
```

16.35 post

```
web3.shh.post(object [, callback])
```

This method should be called, when we want to post whisper a message to the network.

16.35.1 Parameters

1. **`Object` - The post object:**
    - `symKeyID` - `String` (optional): ID of symmetric key for message encryption (Either `symKeyID` or `pubKey` must be present. Can not be both.).
    - `pubKey` - `String` (optional): The public key for message encryption (Either `symKeyID` or `pubKey` must be present. Can not be both.).
    - `sig` - `String` (optional): The ID of the signing key.
    - `ttl` - `Number`: Time-to-live in seconds.
    - `topic` - `String`: 4 Bytes (mandatory when key is symmetric): Message topic.
    - `payload` - `String`: The payload of the message to be encrypted.
    - `padding` - `Number` (optional): Padding (byte array of arbitrary length).
    - `powTime` - `Number` (optional)?: Maximal time in seconds to be spent on proof of work.
    - `powTarget` - `Number` (optional)?: Minimal PoW target required for this message.
    - `targetPeer` - `Number` (optional): Peer ID (for peer-to-peer message only).
2. `callback` - `Function`: (optional) Optional callback, returns an error object as first parameter and the result as second.

16.35.2 Returns

`Promise` returns `Promise` - returns a promise. Upon success, the `then` function will be passed a string representing the hash of the sent message. On error, the `catch` function will be passed a string containing the reason for the error.

16.35.3 Example

```
const identities = [];

Promise.all([
    web3.shh.newSymKey().then((id) => {identities.push(id);}),
    web3.shh.newKeyPair().then((id) => {identities.push(id);})

]).then(() => {

    // will receive also its own message send, below
    const subscription = shh.subscribe("messages", {
        symKeyID: identities[0],
        topics: ['0xffaadd11']
    }).on('data', console.log);
```

(continues on next page)

(continued from previous page)

```
}).then(() => {
   web3.shh.post({
        symKeyID: identities[0], // encrypts using the sym key ID
        sig: identities[1], // signs the message using the keyPair ID
        ttl: 10,
        topic: '0xffaadd11',
        payload: '0xffffffdddddd1122',
        powTime: 3,
        powTarget: 0.5
    }).then(hash => console.log(`Message with hash ${hash} was successfuly sent`))
    .catch(err => console.log("Error: ", err));
});
```

16.36 subscribe

```
web3.shh.subscribe('messages', options [, callback])
```

Subscribe for incoming whisper messages.

16.36.1 Parameters

1. `"messages"` - `String`: Type of the subscription.
2. **`Object` - The subscription options:**
    - `symKeyID` - `String`: ID of symmetric key for message decryption.
    - `privateKeyID` - `String`: ID of private (asymmetric) key for message decryption.
    - `sig` - `String` (optional): Public key of the signature, to verify.
    - `topics`- `Array` (optional when "privateKeyID" key is given): Filters messages by this topic(s). Each topic must be a 4 bytes HEX string.
    - `minPow` - `Number` (optional): Minimal PoW requirement for incoming messages.
    - `allowP2P` - `Boolean` (optional): Indicates if this filter allows processing of direct peer-to-peer messages (which are not to be forwarded any further, because they might be expired). This might be the case in some very rare cases, e.g. if you intend to communicate to MailServers, etc.
3. `callback` - `Function`: (optional) Optional callback, returns an error object as first parameter and the result as second. Will be called for each incoming subscription, and the subscription itself as 3 parameter.

16.36.2 Notification Returns

`Object` - The incoming message:

- `hash` - `String`: Hash of the enveloped message.
- `sig` - `String`: Public key which signed this message.
- `recipientPublicKey` - `String`: The recipients public key.
- `timestamp` - `String`: Unix timestamp of the message genertion.

- `ttl` - `Number`: Time-to-live in seconds.
- `topic` - `String`: 4 Bytes HEX string message topic.
- `payload` - `String`: Decrypted payload.
- `padding` - `Number`: Optional padding (byte array of arbitrary length).
- `pow` - `Number`: Proof of work value.

16.36.3 Example

```
web3.shh.subscribe('messages', {
    symKeyID: 'bf31b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f',
    sig:
↪'0x04d1574d4eab8f3dde4d2dc7ed2c4d699d77cbbdd09167b8fffa099652ce4df00c4c6e0263eafe05007a46fdf0c8d32b
↪',
    ttl: 20,
    topics: ['0xffddaa11'],
    minPow: 0.8,
}, (error, message, subscription) => {

    console.log(message);
    > {
        "hash": "0x4158eb81ad8e30cfcee67f20b1372983d388f1243a96e39f94fd2797b1e9c78e",
        "padding":
↪"0xc15f786f34e5cef0fef6ce7c1185d799ecdb5ebca72b3310648c5588db2e99a0d73301c7a8d90115a91213f0bc9c722
↪",
        "payload": "0xdeadbeaf",
        "pow": 0.5371803278688525,
        "recipientPublicKey": null,
        "sig": null,
        "timestamp": 1496991876,
        "topic": "0x01020304",
        "ttl": 50
    }
})
// or
.on('data', (message) => { ... });
```

16.37 clearSubscriptions

```
web3.shh.clearSubscriptions()
```

Resets subscriptions.

Note: This will not reset subscriptions from other packages like `web3-eth`, as they use their own requestManager.

16.37.1 Parameters

1. `Boolean`: If `true` it keeps the `"syncing"` subscription.

16.37.2 Returns

`Boolean`

16.37.3 Example

```
web3.shh.subscribe('messages', {...} , () => { ... });

...

web3.shh.clearSubscriptions();
```

16.38 newMessageFilter

```
web3.shh.newMessageFilter(options)
```

Create a new filter within the node. This filter can be used to poll for new messages that match the set of criteria.

16.38.1 Parameters

1. `Object`: See *web3.shh.subscribe() options* for details.

16.38.2 Returns

`Promise<string>` - Returns the filter ID.

16.38.3 Example

```
web3.shh.newMessageFilter()
.then(console.log);
> "2b47fbafb3cce24570812a82e6e93cd9e2551bbc4823f6548ff0d82d2206b326"
```

16.39 deleteMessageFilter

```
web3.shh.deleteMessageFilter(id)
```

Deletes a message filter in the node.

16.39.1 Parameters

1. `String`: The filter ID created with `shh.newMessageFilter()`.

16.39.2 Returns

`Promise<boolean>` - Returns `true` on success, error on failure.

16.39.3 Example

```
web3.shh.deleteMessageFilter(
↪'2b47fbafb3cce24570812a82e6e93cd9e2551bbc4823f6548ff0d82d2206b326')
.then(console.log);
> true
```

16.40 getFilterMessages

```
web3.shh.getFilterMessages(id)
```

Retrieve messages that match the filter criteria and are received between the last time this function was called and now.

16.40.1 Parameters

1. `String`: The filter ID created with `shh.newMessageFilter()`.

16.40.2 Returns

`Promise<Array>` - Returns an array of message objects like *web3.shh.subscribe() notification returns*

16.40.3 Example

```
web3.shh.getFilterMessages(
↪'2b47fbafb3cce24570812a82e6e93cd9e2551bbc4823f6548ff0d82d2206b326')
.then(console.log);
> [{
    "hash": "0x4158eb81ad8e30cfcee67f20b1372983d388f1243a96e39f94fd2797b1e9c78e",
    "padding":
↪"0xc15f786f34e5cef0fef6ce7c1185d799ecdb5ebca72b3310648c5588db2e99a0d73301c7a8d90115a91213f0bc9c722
↪",
    "payload": "0xdeadbeaf",
    "pow": 0.5371803278688525,
    "recipientPublicKey": null,
    "sig": null,
    "timestamp": 1496991876,
    "topic": "0x01020304",
    "ttl": 50
},{...}]
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 17

web3.utils

This package provides utility functions for Ethereum dapps and other web3.js packages.

17.1 randomHex

```
web3.utils.randomHex(size)
```

The randomHex library to generate cryptographically strong pseudo-random HEX strings from a given byte size.

17.1.1 Parameters

1. `size` - `Number`: The byte size for the HEX string, e.g. `32` will result in a 32 bytes HEX string with 64 characters preficed with "0x".

17.1.2 Returns

`String`: The generated random HEX string.

17.1.3 Example

```
web3.utils.randomHex(32)
> "0xa5b9d60f32436310afebcfda832817a68921beb782fabf7915cc0460b443116a"

web3.utils.randomHex(4)
> "0x6892ffc6"

web3.utils.randomHex(2)
```

(continues on next page)

(continued from previous page)

```
> "0x99d6"

web3.utils.randomHex(1)
> "0x9a"

web3.utils.randomHex(0)
> "0x"
```

17.2 BN

```
web3.utils.BN(mixed)
```

The BN.js library for calculating with big numbers in JavaScript. See the BN.js documentation for details.

Note: For safe conversion of many types, incl BigNumber.js use *utils.toBN*

17.2.1 Parameters

1. `value` - `String|Number`: A number, number string or HEX string to convert to a BN object.

17.2.2 Returns

`Object`: The BN.js instance.

17.2.3 Example

```
const BN = web3.utils.BN;

new BN(1234).toString();
> "1234"

new BN('1234').add(new BN('1')).toString();
> "1235"

new BN('0xea').toString();
> "234"
```

17.3 isBN

```
web3.utils.isBN(bn)
```

Checks if a given value is a BN.js instance.

17.3.1 Parameters

1. `bn` - `Object`: An BN.js instance.

17.3.2 Returns

`Boolean`

17.3.3 Example

```
const number = new BN(10);

web3.utils.isBN(number);
> true
```

17.4 isBigNumber

```
web3.utils.isBigNumber(bignumber)
```

Checks if a given value is a BigNumber.js instance.

17.4.1 Parameters

1. `BigNumber` - `Object`: A BigNumber.js instance.

17.4.2 Returns

`Boolean`

17.4.3 Example

```
const number = new BigNumber(10);

web3.utils.isBigNumber(number);
> true
```

17.5 keccak256

```
web3.utils.keccak256(string)
web3.utils.sha3(string) // ALIAS
```

Will calculate the keccak256 of the input.

Note: To mimic the keccak256 behaviour of solidity use *solidtySha3*

17.5.1 Parameters

1. `string` - `String`: A string to hash.

17.5.2 Returns

`String`: the result hash.

17.5.3 Example

```
web3.utils.keccak256('234'); // taken as string
> "0xc1912fee45d61c87cc5ea59dae311904cd86b84fee17cc96966216f811ce6a79"

web3.utils.keccak256(new BN('234'));
> "0xbc36789e7a1e281436464229828f817d6612f7b477d66591ff96a9e064bcc98a"

web3.utils.keccak256(234);
> null // can't calculate the hash of a number

web3.utils.keccak256(0xea); // same as above, just the HEX representation of the␣
↪number
> null

web3.utils.keccak256('0xea'); // will be converted to a byte array first, and then␣
↪hashed
> "0x2f20677459120677484f7104c76deb6846a2c071f9b3152c103bb12cd54d1a4a"
```

17.6 solidtySha3

```
web3.utils.solidtySha3(param1 [, param2, ...])
```

Will calculate the sha3 of given input parameters in the same way solidity would. This means arguments will be ABI converted and tightly packed before being hashed.

17.6.1 Parameters

1. `paramX` - `Mixed`: Any type, or an object with `{type: 'uint', value: '123456'}` or `{t: 'bytes', v: '0xfff456'}`. Basic types are autodetected as follows:
   - `String` non numerical UTF-8 string is interpreted as `string`.
   - `String|Number|BN|HEX` positive number is interpreted as `uint256`.
   - `String|Number|BN` negative number is interpreted as `int256`.

- `Boolean` as `bool`.
- `String` HEX string with leading `0x` is interpreted as `bytes`.
- `HEX` HEX number representation is interpreted as `uint256`.

17.6.2 Returns

`String`: the result hash.

17.6.3 Example

```
web3.utils.soliditySha3('234564535', '0xfff23243', true, -10);
// auto detects:        uint256,      bytes,     bool,   int256
> "0x3e27a893dc40ef8a7f0841d96639de2f58a132be5ae466d40087a2cfa83b7179"


web3.utils.soliditySha3('Hello!%'); // auto detects: string
> "0x661136a4267dba9ccdf6bfddb7c00e714de936674c4bdb065a531cf1cb15c7fc"


web3.utils.soliditySha3('234'); // auto detects: uint256
> "0x61c831beab28d67d1bb40b5ae1a11e2757fa842f031a2d0bc94a7867bc5d26c2"

web3.utils.soliditySha3(0xea); // same as above
> "0x61c831beab28d67d1bb40b5ae1a11e2757fa842f031a2d0bc94a7867bc5d26c2"

web3.utils.soliditySha3(new BN('234')); // same as above
> "0x61c831beab28d67d1bb40b5ae1a11e2757fa842f031a2d0bc94a7867bc5d26c2"

web3.utils.soliditySha3({type: 'uint256', value: '234'}); // same as above
> "0x61c831beab28d67d1bb40b5ae1a11e2757fa842f031a2d0bc94a7867bc5d26c2"

web3.utils.soliditySha3({t: 'uint', v: new BN('234')}); // same as above
> "0x61c831beab28d67d1bb40b5ae1a11e2757fa842f031a2d0bc94a7867bc5d26c2"


web3.utils.soliditySha3('0x407D73d8a49eeb85D32Cf465507dd71d507100c1');
> "0x4e8ebbefa452077428f93c9520d3edd60594ff452a29ac7d2ccc11d47f3ab95b"

web3.utils.soliditySha3({t: 'bytes', v: '0x407D73d8a49eeb85D32Cf465507dd71d507100c1'}
↪);
> "0x4e8ebbefa452077428f93c9520d3edd60594ff452a29ac7d2ccc11d47f3ab95b" // same result␣
↪as above


web3.utils.soliditySha3({t: 'address', v: '0x407D73d8a49eeb85D32Cf465507dd71d507100c1
↪'});
> "0x4e8ebbefa452077428f93c9520d3edd60594ff452a29ac7d2ccc11d47f3ab95b" // same as␣
↪above, but will do a checksum check, if its multi case


web3.utils.soliditySha3({t: 'bytes32', v: '0x407D73d8a49eeb85D32Cf465507dd71d507100c1
↪'});
> "0x3c69a194aaf415ba5d6afca734660d0a3d45acdc05d54cd1ca89a8988e7625b4" // different␣
↪result as above
```

(continues on next page)

(continued from previous page)

```
web3.utils.solidity Sha3({t: 'string', v: 'Hello!%'}, {t: 'int8', v:-23}, {t: 'address
↪', v: '0x85F43D8a49eeB85d32Cf465507DD71d507100C1d'});
> "0xa13b31627c1ed7aaded5aecec71baf02fe123797fffd45e662eac8e06fbe4955"
```

17.7 isHex

```
web3.utils.isHex(hex)
```

Checks if a given string is a HEX string.

17.7.1 Parameters

1. `hex` - `String|HEX`: The given HEX string.

17.7.2 Returns

`Boolean`

17.7.3 Example

```
web3.utils.isHex('0xc1912');
> true

web3.utils.isHex(0xc1912);
> true

web3.utils.isHex('c1912');
> true

web3.utils.isHex(345);
> true // this is tricky, as 345 can be a a HEX representation or a number, be␣
↪careful when not having a 0x in front!

web3.utils.isHex('0xZ1912');
> false

web3.utils.isHex('Hello');
> false
```

17.8 isHexStrict

```
web3.utils.isHexStrict(hex)
```

Checks if a given string is a HEX string. Difference to `web3.utils.isHex()` is that it expects HEX to be prefixed with `0x`.

17.8.1 Parameters

1. `hex` - `String|HEX`: The given HEX string.

17.8.2 Returns

`Boolean`

17.8.3 Example

```
web3.utils.isHexStrict('0xc1912');
> true

web3.utils.isHexStrict(0xc1912);
> false

web3.utils.isHexStrict('c1912');
> false

web3.utils.isHexStrict(345);
> false // this is tricky, as 345 can be a a HEX representation or a number, be␣
↪careful when not having a 0x in front!

web3.utils.isHexStrict('0xZ1912');
> false

web3.utils.isHex('Hello');
> false
```

17.9 isAddress

```
web3.utils.isAddress(address,  [, chainId])
```

Checks if a given string is a valid Ethereum address. It will also check the checksum, if the address has upper and lowercase letters.

17.9.1 Parameters

1. `address` - `String`: An address string.
2. `chainId` - `number` (optional): Chain id where checksummed address should be valid, defaults to `null`. RSKIP-60 <https://github.com/rsksmart/RSKIPs/blob/master/IPs/RSKIP60.md> for details.

17.9.2 Returns

`Boolean`

17.9.3 Example

```
web3.utils.isAddress('0xc1912fee45d61c87cc5ea59dae31190fffff232d');
> true

web3.utils.isAddress('c1912fee45d61c87cc5ea59dae31190fffff232d');
> true

web3.utils.isAddress('0XC1912FEE45D61C87CC5EA59DAE31190FFFFF232D');
> true // as all is uppercase, no checksum will be checked

web3.utils.isAddress('0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d');
> true

web3.utils.isAddress('0xC1912fEE45d61C87Cc5EA59DaE31190FFFFf232d');
> false // wrong checksum

web3.utils.isAddress('0x5aaEB6053f3e94c9b9a09f33669435E7ef1bEAeD', 30);
> true
```

17.10 toChecksumAddress

```
web3.utils.toChecksumAddress(address[, chainId])
```

Will convert an upper or lowercase Ethereum address to a checksum address.

17.10.1 Parameters

1. `address` - `String`: An address string.
2. `chainId` - `number` (optional): Chain id where checksummed address should be valid, defaults to `null`. RSKIP-60 <https://github.com/rsksmart/RSKIPs/blob/master/IPs/RSKIP60.md> for details.

17.10.2 Returns

`String`: The checksum address.

17.10.3 Example

```
web3.utils.toChecksumAddress('0xc1912fee45d61c87cc5ea59dae31190fffff232d');
> "0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d"

web3.utils.toChecksumAddress('0XC1912FEE45D61C87CC5EA59DAE31190FFFFF232D');
> "0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d" // same as above
```

(continues on next page)

(continued from previous page)

```
web3.utils.toChecksumAddress('0x5aaeb6053f3e94c9b9a09f33669435e7ef1beaed', 30);
> "0x5aaEB6053f3e94c9b9a09f33669435E7ef1bEAeD"
```

17.11 stripHexPrefix

Removes the prefix `0x` from a given hex if it exists.

17.11.1 Parameters

1. `hex` - `String`: Hex

17.11.2 Returns

`String`: Hex without prefix.

17.11.3 Example

17.12 checkAddressChecksum

```
web3.utils.checkAddressChecksum(address [, chainId])
```

Checks the checksum of a given address. Will also return false on non-checksum addresses.

17.12.1 Parameters

1. `address` - `String`: An address string.
2. `chainId` - `number` (optional): Chain id where checksummed address should be valid, defaults to `null`. RSKIP-60 <https://github.com/rsksmart/RSKIPs/blob/master/IPs/RSKIP60.md> for details.

17.12.2 Returns

`Boolean`: `true` when the checksum of the address is valid, `false` if its not a checksum address, or the checksum is invalid.

17.12.3 Example

```
web3.utils.checkAddressChecksum('0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d');
> true

web3.utils.checkAddressChecksum('0x5aAeb6053F3e94c9b9A09F33669435E7EF1BEaEd', 31);
> true
```

17.13 toHex

```
web3.utils.toHex(mixed)
```

Will auto convert any given value to HEX. Number strings will interpreted as numbers. Text strings will be interpreted as UTF-8 strings.

17.13.1 Parameters

1. `value` - `String|Number|BN|BigNumber`: The input to convert to HEX.

17.13.2 Returns

`String`: The resulting HEX string.

17.13.3 Example

```
web3.utils.toHex('234');
> "0xea"

web3.utils.toHex(234);
> "0xea"

web3.utils.toHex(new BN('234'));
> "0xea"

web3.utils.toHex(new BigNumber('234'));
> "0xea"

web3.utils.toHex('I have 100€');
> "0x4920686176652031303030e282ac"
```

17.14 toBN

```
web3.utils.toBN(number)
```

Will safely convert any given value (including BigNumber.js instances) into a BN.js instance, for handling big numbers in JavaScript.

Note: For just the BN.js class use *utils.BN*

17.14.1 Parameters

1. `number` - `String|Number|HEX`: Number to convert to a big number.

17.14.2 Returns

`Object`: The BN.js instance.

17.14.3 Example

```
web3.utils.toBN(1234).toString();
> "1234"

web3.utils.toBN('1234').add(web3.utils.toBN('1')).toString();
> "1235"

web3.utils.toBN('0xea').toString();
> "234"
```

17.15 hexToNumberString

```
web3.utils.hexToNumberString(hex)
```

Returns the number representation of a given HEX value as a string.

17.15.1 Parameters

1. `hexString` - `String|HEX`: A string to hash.

17.15.2 Returns

`String`: The number as a string.

17.15.3 Example

```
web3.utils.hexToNumberString('0xea');
> "234"
```

17.16 hexToNumber

```
web3.utils.hexToNumber(hex)
web3.utils.toDecimal(hex) // ALIAS, deprecated
```

Returns the number representation of a given HEX value.

Note: This is not useful for big numbers, rather use *utils.toBN* instead.

17.16.1 Parameters

1. `hexString` - `String|HEX`: A string to hash.

17.16.2 Returns

`Number`

17.16.3 Example

```
web3.utils.hexToNumber('0xea');
> 234
```

17.17 numberToHex

```
web3.utils.numberToHex(number)
web3.utils.fromDecimal(number) // ALIAS, deprecated
```

Returns the HEX representation of a given number value.

17.17.1 Parameters

1. `number` - `String|Number|BN|BigNumber`: A number as string or number.

17.17.2 Returns

`String`: The HEX value of the given number.

17.17.3 Example

```
web3.utils.numberToHex('234');
> '0xea'
```

17.18 hexToUtf8

```
web3.utils.hexToUtf8(hex)
web3.utils.hexToString(hex) // ALIAS
web3.utils.toUtf8(hex) // ALIAS, deprecated
```

Returns the UTF-8 string representation of a given HEX value.

17.18.1 Parameters

1. `hex` - `String`: A HEX string to convert to a UTF-8 string.

17.18.2 Returns

`String`: The UTF-8 string.

17.18.3 Example

```
web3.utils.hexToUtf8('0x4920686176652031303e282ac');
> "I have 100€"
```

17.19 hexToAscii

```
web3.utils.hexToAscii(hex)
web3.utils.toAscii(hex) // ALIAS, deprecated
```

Returns the ASCII string representation of a given HEX value.

17.19.1 Parameters

1. `hex` - `String`: A HEX string to convert to a ASCII string.

17.19.2 Returns

`String`: The ASCII string.

17.19.3 Example

```
web3.utils.hexToAscii('0x4920686176652031303021');
> "I have 100!"
```

17.20 utf8ToHex

```
web3.utils.utf8ToHex(string)
web3.utils.stringToHex(string) // ALIAS
web3.utils.fromUtf8(string) // ALIAS, deprecated
```

Returns the HEX representation of a given UTF-8 string.

17.20.1 Parameters

1. `string` - `String`: A UTF-8 string to convert to a HEX string.

17.20.2 Returns

`String`: The HEX string.

17.20.3 Example

```
web3.utils.utf8ToHex('I have 100€');
> "0x4920686176652031303030e282ac"
```

17.21 asciiToHex

```
web3.utils.asciiToHex(string)
web3.utils.fromAscii(string) // ALIAS, deprecated
```

Returns the HEX representation of a given ASCII string. If you would like to transform an ASCII string into a valid `bytes4`, `bytes8` etc. value then please pass the correct length as the second parameter.

17.21.1 Parameters

1. `string` - `String`: A ASCII string to convert to a HEX string.
2. `length` - `Number`: The length of the returned hex string. The default size is `32` e.g.: `bytes32`.

17.21.2 Returns

`String`: The HEX string.

17.21.3 Example

```
web3.utils.asciiToHex('I have 100!');
> "0x4920686176652031303021000000000000000000000000000000000000000000"

// transforming to a bytes4 value:
web3.utils.asciiToHex('yes', 4);

// transforming to a bytes8 value:
web3.utils.asciiToHex('yes', 8);

//etc.
```

17.22 hexToBytes

```
web3.utils.hexToBytes(hex)
```

Returns a byte array from the given HEX string.

17.22.1 Parameters

1. `hex` - `String|HEX`: A HEX to convert.

17.22.2 Returns

`Array`: The byte array.

17.22.3 Example

```
web3.utils.hexToBytes('0x000000ea');
> [ 0, 0, 0, 234 ]

web3.utils.hexToBytes(0x000000ea);
> [ 234 ]
```

17.23 bytesToHex

```
web3.utils.bytesToHex(byteArray)
```

Returns a HEX string from a byte array.

17.23.1 Parameters

1. `byteArray` - `Array`: A byte array to convert.

17.23.2 Returns

`String`: The HEX string.

17.23.3 Example

```
web3.utils.bytesToHex([ 72, 101, 108, 108, 111, 33, 36 ]);
> "0x48656c6c6f2125"
```

17.24 toWei

```
web3.utils.toWei(number [, unit])
```

Converts any ether value value into wei.

Note: "wei" are the smallest ethere unit, and you should always make calculations in wei and convert only for display reasons.

17.24.1 Parameters

1. `number` - `String|BN`: The value.
2. **`unit` - `String` (optional, defaults to `"ether"`): The ether to convert from. Possible units are:**
    - `noether`: '0'
    - `wei`: '1'
    - `kwei`: '1000'
    - `Kwei`: '1000'
    - `babbage`: '1000'
    - `femtoether`: '1000'
    - `mwei`: '1000000'
    - `Mwei`: '1000000'
    - `lovelace`: '1000000'
    - `picoether`: '1000000'
    - `gwei`: '1000000000'
    - `Gwei`: '1000000000'
    - `shannon`: '1000000000'
    - `nanoether`: '1000000000'
    - `nano`: '1000000000'
    - `szabo`: '1000000000000'

- `microether`: '1000000000000'
- `micro`: '1000000000000'
- `finney`: '1000000000000000'
- `milliether`: '1000000000000000'
- `milli`: '1000000000000000'
- `ether`: '1000000000000000000'
- `kether`: '1000000000000000000000'
- `grand`: '1000000000000000000000'
- `mether`: '1000000000000000000000000'
- `gether`: '1000000000000000000000000000'
- `tether`: '1000000000000000000000000000000'

17.24.2 Returns

`String|BN`: If a string is given it returns a number string, otherwise a BN.js instance.

17.24.3 Example

```
web3.utils.toWei('1', 'ether');
> "1000000000000000000"

web3.utils.toWei('1', 'finney');
> "1000000000000000"

web3.utils.toWei('1', 'szabo');
> "1000000000000"

web3.utils.toWei('1', 'shannon');
> "1000000000"
```

17.25 fromWei

```
web3.utils.fromWei(number [, unit])
```

Converts any wei value into a ether value.

Note: "wei" are the smallest ethere unit, and you should always make calculations in wei and convert only for display reasons.

17.25.1 Parameters

1. `number` - `String|BN`: The value in wei.
2. **`unit` - `String` (optional, defaults to `"ether"`): The ether to convert to. Possible units are:**
    - `noether`: '0'
    - `wei`: '1'
    - `kwei`: '1000'
    - `Kwei`: '1000'
    - `babbage`: '1000'
    - `femtoether`: '1000'
    - `mwei`: '1000000'
    - `Mwei`: '1000000'
    - `lovelace`: '1000000'
    - `picoether`: '1000000'
    - `gwei`: '1000000000'
    - `Gwei`: '1000000000'
    - `shannon`: '1000000000'
    - `nanoether`: '1000000000'
    - `nano`: '1000000000'
    - `szabo`: '1000000000000'
    - `microether`: '1000000000000'
    - `micro`: '1000000000000'
    - `finney`: '1000000000000000'
    - `milliether`: '1000000000000000'
    - `milli`: '1000000000000000'
    - `ether`: '1000000000000000000'
    - `kether`: '1000000000000000000000'
    - `grand`: '1000000000000000000000'
    - `mether`: '1000000000000000000000000'
    - `gether`: '1000000000000000000000000000'
    - `tether`: '1000000000000000000000000000000'

17.25.2 Returns

`String`: It always returns a string number.

17.25.3 Example

```
web3.utils.fromWei('1', 'ether');
> "0.000000000000000001"

web3.utils.fromWei('1', 'finney');
> "0.000000000000001"

web3.utils.fromWei('1', 'szabo');
> "0.000000000001"

web3.utils.fromWei('1', 'shannon');
> "0.000000001"
```

17.26 unitMap

```
web3.utils.unitMap
```

Shows all possible ether value and their amount in wei.

17.26.1 Return value

- **`Object` with the following properties:**
  - `noether`: '0'
  - `wei`: '1'
  - `kwei`: '1000'
  - `Kwei`: '1000'
  - `babbage`: '1000'
  - `femtoether`: '1000'
  - `mwei`: '1000000'
  - `Mwei`: '1000000'
  - `lovelace`: '1000000'
  - `picoether`: '1000000'
  - `gwei`: '1000000000'
  - `Gwei`: '1000000000'
  - `shannon`: '1000000000'
  - `nanoether`: '1000000000'
  - `nano`: '1000000000'
  - `szabo`: '1000000000000'
  - `microether`: '1000000000000'
  - `micro`: '1000000000000'

- finney: '1000000000000000'
- milliether: '1000000000000000'
- milli: '1000000000000000'
- ether: '1000000000000000000'
- kether: '1000000000000000000000'
- grand: '1000000000000000000000'
- mether: '1000000000000000000000000'
- gether: '1000000000000000000000000000'
- tether: '1000000000000000000000000000000'

17.26.2 Example

```
web3.utils.unitMap
> {
    noether: '0',
    wei:         '1',
    kwei:        '1000',
    Kwei:        '1000',
    babbage:     '1000',
    femtoether: '1000',
    mwei:        '1000000',
    Mwei:        '1000000',
    lovelace:    '1000000',
    picoether:   '1000000',
    gwei:        '1000000000',
    Gwei:        '1000000000',
    shannon:     '1000000000',
    nanoether:   '1000000000',
    nano:        '1000000000',
    szabo:       '1000000000000',
    microether: '1000000000000',
    micro:       '1000000000000',
    finney:      '1000000000000000',
    milliether: '1000000000000000',
    milli:       '1000000000000000',
    ether:       '1000000000000000000',
    kether:      '1000000000000000000000',
    grand:       '1000000000000000000000',
    mether:      '1000000000000000000000000',
    gether:      '1000000000000000000000000000',
    tether:      '1000000000000000000000000000000'
}
```

17.27 padLeft

```
web3.utils.padLeft(string, characterAmount [, sign])
web3.utils.leftPad(string, characterAmount [, sign]) // ALIAS
```

Adds a padding on the left of a string, Useful for adding paddings to HEX strings.

17.27.1 Parameters

1. `string` - `String`: The string to add padding on the left.
2. `characterAmount` - `Number`: The number of characters the total string should have.
3. `sign` - `String` (optional): The character sign to use, defaults to `"0"`.

17.27.2 Returns

`String`: The padded string.

17.27.3 Example

```
web3.utils.padLeft('0x3456ff', 20);
> "0x00000000000003456ff"

web3.utils.padLeft(0x3456ff, 20);
> "0x00000000000003456ff"

web3.utils.padLeft('Hello', 20, 'x');
> "xxxxxxxxxxxxxxxHello"
```

17.28 padRight

```
web3.utils.padRight(string, characterAmount [, sign])
web3.utils.rightPad(string, characterAmount [, sign]) // ALIAS
```

Adds a padding on the right of a string, Useful for adding paddings to HEX strings.

17.28.1 Parameters

1. `string` - `String`: The string to add padding on the right.
2. `characterAmount` - `Number`: The number of characters the total string should have.
3. `sign` - `String` (optional): The character sign to use, defaults to `"0"`.

17.28.2 Returns

`String`: The padded string.

17.28.3 Example

```
web3.utils.padRight('0x3456ff', 20);
> "0x3456ff00000000000000"

web3.utils.padRight(0x3456ff, 20);
> "0x3456ff00000000000000"

web3.utils.padRight('Hello', 20, 'x');
> "Helloxxxxxxxxxxxxxxx"
```

17.29 toTwosComplement

```
web3.utils.toTwosComplement(number)
```

Converts a negative numer into a two's complement.

17.29.1 Parameters

1. `number` - `Number|String|BigNumber`: The number to convert.

17.29.2 Returns

`String`: The converted hex string.

17.29.3 Example

```
web3.utils.toTwosComplement('-1');
> "0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"

web3.utils.toTwosComplement(-1);
> "0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"

web3.utils.toTwosComplement('0x1');
> "0x0000000000000000000000000000000000000000000000000000000000000001"

web3.utils.toTwosComplement(-15);
> "0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff1"

web3.utils.toTwosComplement('-0x1');
> "0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"
```

17.30 getSignatureParameters

```
web3.utils.getSignatureParameters(string)
```

Gets the r, s and v values of an ECDSA signature

17.30.1 Parameters

1. `string` - `String`: An ECDSA signature.

17.30.2 Returns

`Object`: Object containing r,s,v values.

17.30.3 Example

```
web3.utils.getSignatureParameters(
↪'0x5763ab346198e3e6cc4d53996ccdeca0c941cb6cb70d671d97711c421d3bf7922c77ef244ad40e5262d1721bf9638fb0
↪');
> "{ r: '0x5763ab346198e3e6cc4d53996ccdeca0c941cb6cb70d671d97711c421d3bf792', s:
↪'0x2c77ef244ad40e5262d1721bf9638fb06bab8ed3c43bfaa80d6da0be9bbd33dc', v: 27 }"
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 18

Module API

The `Module API` gives you the possibility to create your **own custom Web3 Module** with JSON-RPC methods, subscriptions, or contracts. The provided modules from the Web3 library are also written with the `Module API` the core does provide.

The goal of the `Module API` is to provide the possibility to extend and customize the JSON-RPC methods, contracts, and subscriptions to project specific classes with a similar kind of API the DApp developer knows from the Web3.js library. It's possible with the Web3 Module API to create complex contract APIs and tools for the development of a DApp.

These are the core modules which are providing all the classes for the Web3 Module API.

- *web3-core*
- *web3-core-method*
- *web3-core-subscriptions*
- *Contract*

18.1 Example

```
import * as Utils from 'web3-utils';
import {formatters} from 'web3-core-helpers';
import {AbstractWeb3Module} from 'web3-core';
import {AbstractMethodFactory, GetBlockByNumberMethod, AbstractMethod} from 'web3-
↪core-method';

class MethodFactory extends AbstractMethodFactory {
    /**
     * @param {Utils} utils
     * @param {Object} formatters
     *
     * @constructor
     */
```

(continues on next page)

(continued from previous page)

```
    constructor(utils, formatters) {
        super(utils, formatters);

        this.methods = {
            getBlockByNumber: GetBlockByNumberMethod
        };
    }
}

class Example extends AbstractWeb3Module {
    /**
     * @param {AbstractSocketProvider|HttpProvider|CustomProvider|String} provider
     * @param {Web3ModuleOptions} options
     * @param {Net.Socket} net
     *
     * @constructor
     */
    constructor(provider, net, options) {
        super(provider, net, new MethodFactory(Utils, formatters), options;
    }

    /**
     * Executes the eth_sign JSON-RPC method
     *
     * @method sign
     *
     * @returns {Promise}
     */
    sign() {
        const method = new AbstractMethod('eth_sign', 2, Utils, formatters, this);
        method.setArguments(arguments)

        return method.execute();
    }

    /**
     * Executes the eth_subscribe JSON-RPC method with the subscription type logs
     *
     * @method logs
     *
     * @returns {LogSubscription}
     */
    logs(options) {
        return new LogSubscription(
          options,
          Utils,
          formatters,
          this,
          new GetPastLogsMethod(Utils, formatters, this)
        );
    }
}

const example = new Example(provider, net, options);

example.sign('0x0', 'message').then(console.log);
// > "response"
```

(continues on next page)

(continued from previous page)

```
example.sign('0x0', 'message', (error, response) => {
    console.log(response);
};
// > "response"

const block = example.getBlockByNumber(1).then(console.log);
// > {}

example.logs(options).subscribe(console.log);
> {}
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 19

Contract Module API

The `Contract Module API` does provide to possibility to create project specific contracts with pre-injecting of the ABI or customizing of the default behaviour of a Web3 contract.

19.1 Contract

The exported class `Contract` is here to simply pre-inject a contract ABI.

19.1.1 Parameters

1. `provider - AbstractSocketProvider | HttpProvider | CustomProvider | String`: A Web3.js provider.
2. `abi - Array`: Contract ABI
3. `accounts` - *Accounts*
4. `options - Web3ModuleOptions`

19.1.2 Example

```
import {MyABI, options} from '../folder/file.js';
import {Accounts} from 'web3-eth-accounts';
import {Contract} from 'web3-eth-contract';

export class MyContract extends Contract {
    constructor(provider) {
        super(provider, MyAbi, new Accounts(...), '0x0', options);
    }
}
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 20

Core Module

The `Core Module` does provide the `AbstractWeb3Module` to implement Web3 compatible modules.

20.1 AbstractWeb3Module

Source: AbstractWeb3Module

The `AbstractWeb3Module` does have the following constructor parameters:

- `provider - AbstractSocketProvider | HttpProvider | CustomProvider | String` The provider class or string.
- `options - Web3ModuleOptions` These are the default `options` of a Web3 module. (optional)
- `methodFactory - AbstractMethodFactory` The *AbstractMethodFactory* will be used in the module proxy for the JSON-RPC method calls. (optional)
- `net - net.Socket` The `net.Socket` object of the NodeJS net module. (optional)

20.1.1 Example

```
import {AbstractWeb3Module} from 'web3-core';

class Example extends AbstractWeb3Module {
    /**
     * @param {AbstractSocketProvider|HttpProvider|CustomProvider|String} provider
     * @param {AbstractMethodFactory} methodFactory
     * @param {Web3ModuleOptions} options
     * @param {Net.Socket} nodeNet
     *
     * @constructor
     */
    constructor(provider, net, methodFactory, options) {
```

(continues on next page)

(continued from previous page)

```
        super(provider, net, methodFactory, options;
    }
}
```

Interface of the `AbstractWeb3Module` class:

20.2 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

20.2.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

20.2.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

20.3 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

20.3.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block
- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

20.4 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

20.4.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

20.5 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

20.5.1 Returns

`string|number`: The current value of the defaultGasPrice property.

20.6 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

20.6.1 Returns

`string|number`: The current value of the defaultGas property.

20.7 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

20.7.1 Returns

`number`: The current value of transactionBlockTimeout

20.8 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

20.8.1 Returns

`number`: The current value of transactionConfirmationBlocks

20.9 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

20.9.1 Returns

`number`: The current value of transactionPollingTimeout

20.10 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

20.10.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

20.11 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

20.11.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

20.11.2 Returns

`Boolean`

20.11.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

20.12 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

20.12.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

20.12.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

20.13 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

20.13.1 Returns

`Object`: The given provider set or `false`.

20.13.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

20.14 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

20.14.1 Returns

`Object`: The current provider set.

20.14.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

20.15 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

20.15.1 Parameters

none

20.15.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

20.15.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 21

Core Method Module

The `Core Method Module` does provide all method classes and the abstract method factory which will be used in the `AbstractWeb3Module`.

21.1 AbstractMethodFactory

Source: AbstractMethodFactory

The `AbstractMethodFactory` does have the following constructor parameters:

- `utils` - `Utils` The `Utils` object from the `web3-utils` module.
- `formatters` - `Object` The formatters object from the `web3-core-helpers` module.

21.1.1 Example

```
import {
    AbstractMethodFactory,
    GetBlockByNumberMethod,
    ListeningMethod,
    PeerCountMethod,
    VersionMethod
} from 'web3-core-method';

class MethodFactory extends AbstractMethodFactory {
    /**
     * @param {Utils} utils
     * @param {Object} formatters
     *
     * @constructor
     */
    constructor(utils, formatters) {
```

(continues on next page)

(continued from previous page)

```
        super(utils, formatters);

        this.methods = {
            getId: VersionMethod,
            getBlockByNumber: GetBlockByNumberMethod,
            isListening: ListeningMethod,
            getPeerCount: PeerCountMethod
        };
    }
}
```

21.2 AbstractMethod

Source: AbstractMethod

Because we are always adding new JSON-RPC methods do we just link the methods folder as resource.

Source: Methods

The provided method classes do have the following interface:

The `AbstractMethod` class does have the following constructor parameters:

- `rpcMethod` - `String` The JSON-RPC method name.
- `parametersAmount` - `Number` The amount of parameters this JSON-RPC method has.
- `utils` - `Utils`
- `formatters` - `Object` The formatters object.
- `moduleInstance` - `AbstractWeb3Module`

The `AbstractMethod` class is the base JSON-RPC method class and does provide the basic methods and properties for creating a Web3.js compatible JSON-RPC method.

You're able to overwrite these methods:

- execute(): PromiEvent
- afterExecution(response: any): void
- beforeExecution(moduleInstance: AbstractWeb3Module): void
- setArguments(arguments: IArguments): void
- getArguments(arguments: IArguments): {parameters: any[], callback: Function}

This example will show the usage of the `setArguments(arguments: IArguments)` method.

It's also possible to set the parameters and callback method directly over the `parameters` and `callback` property of the method class.

21.2.1 Example

```
class Example extends AbstractWeb3Module {
    constructor(...) {
        // ...
    }

    sign() {
        const method = new AbstractMethod('eth_sign', 2, utils, formatters, this);
        method.setArguments(arguments)

        return method.execute();
    }
}

const example = new Example(...);

const response = await example.sign('0x0', 'message').
// > "response"


example.sign('0x0', 'message', (error, response) => {
    console.log(response);
};
// > "response"
```

The `AbstractMethod` class interface:

21.3 Type

The static `readonly` property `Type` will be used in the `AbstractMethodFactory` class to determine how the class should get initiated.

Reserved types:

- `observed-transaction-method` - `AbstractObservedTransactionMethod`
- `eth-send-transaction-method` - `EthSendTransactionMethod`

21.3.1 Returns

`string` - Example: `observed-transaction-method`

21.4 beforeExecution

```
method.beforeExecution(moduleInstance)
```

This method will be executed before the JSON-RPC request. It provides the possibility to customize the given parameters or other properties of the current method.

21.4.1 Parameters

- `moduleInstance` - `AbstractWeb3Module` The current `AbstractWeb3Module`.

21.5 afterExecution

```
method.afterExecution(response)
```

This method will get executed when the provider returns with the response. The `afterExecution` method does provide us the possibility to map the response to the desired value.

21.5.1 Parameters

- `response` - `any` The response from the provider.

21.5.2 Returns

`any`

21.6 execute

```
method.execute()
```

This method will execute the current method.

21.6.1 Returns

`Promise<Object|string>|PromiEvent|string`

21.7 rpcMethod

```
method.rpcMethod
```

This property will return the `rpcMethod` string. It will be used for the creation of the JSON-RPC payload object.

21.7.1 Returns

`string`

21.8 parametersAmount

```
method.parametersAmount
```

This property will return the `parametersAmount`. It will be used for validating the given parameters length and for the detection of the callback method.

21.8.1 Returns

`number`

21.9 parameters

```
method.parameters
```

This property does contain the given `parameters`.

Use the `setArguments()` method for setting the parameters and the callback method with the given `IArguments` object.

21.9.1 Returns

`any[]`

21.10 callback

```
method.callback
```

This property does contain the given `callback`.

Use the `setArguments()` method for setting the parameters and the callback method with the given `IArguments` object.

21.10.1 Returns

`undefined`

21.11 setArguments

```
method.setArguments(arguments)
```

This method will be used to set the given method arguments. The `setArguments` method will set the `parameters` and `callback` property.

21.11.1 Parameters

- `arguments` - `Array`: The `arguments` of the function call.

21.11.2 Returns

`Object`

21.12 getArguments

```
method.getArguments()
```

This method will be used to get the method arguments. The `getArguments` method will return a object with the properties `parameters` and `callback`.

21.12.1 Returns

`Object`

21.13 isHash

```
method.isHash(value)
```

This method will check if the given value is a string and starts with `0x`. It will be used in several methods for deciding which JSON-RPC method should get executed.

21.13.1 Parameters

- `value` - `string`

21.13.2 Returns

`boolean`

21.14 AbstractObservedTransactionMethod

Source: AbstractObservedTransactionMethod

The `AbstractObservedTransactionMethod` extends from the AbstractMethod <web3-module-abstract-method and does have the following constructor parameters:

- `rpcMethod` - `String` The JSON-RPC method name.
- `parametersAmount` - `Number` The amount of parameters this JSON-RPC method has.
- `utils` - `Object` The Utils object.
- `formatters` - `Object` The formatters object.
- `transactionObserver` - `TransactionObserver` The `TransactionObserver` class which defines the confirmation process of the transaction.

The `AbstractObservedTransactionMethod` is the base method class for all "send transaction" methods.

Abstract methods:

- afterExecution
- beforeExecution

21.15 Type

The static `readonly` property `Type` will be used in the `AbstractMethodFactory` class to determine how the class should get initiated.

Reserved types:

- `observed-transaction-method` - `AbstractObservedTransactionMethod`
- `eth-send-transaction-method` - `EthSendTransactionMethod`

21.15.1 Returns

`string` - Example: `observed-transaction-method`

21.16 beforeExecution

```
method.beforeExecution(moduleInstance)
```

This method will be executed before the JSON-RPC request. It provides the possibility to customize the given parameters or other properties of the current method.

21.16.1 Parameters

- `moduleInstance` - `AbstractWeb3Module` The current `AbstractWeb3Module`.

21.17 afterExecution

```
method.afterExecution(response)
```

This method will get executed when the provider returns with the response. The `afterExecution` method does provide us the possibility to map the response to the desired value.

21.17.1 Parameters

- `response` - `any` The response from the provider.

21.17.2 Returns

`any`

21.18 execute

```
method.execute()
```

This method will execute the current method.

21.18.1 Returns

`Promise<Object|string>|PromiEvent|string`

21.19 rpcMethod

```
method.rpcMethod
```

This property will return the `rpcMethod` string. It will be used for the creation of the JSON-RPC payload object.

21.19.1 Returns

`string`

21.20 parametersAmount

```
method.parametersAmount
```

This property will return the `parametersAmount`. It will be used for validating the given parameters length and for the detection of the callback method.

21.20.1 Returns

`number`

21.21 parameters

```
method.parameters
```

This property does contain the given `parameters`.

Use the `setArguments()` method for setting the parameters and the callback method with the given `IArguments` object.

21.21.1 Returns

`any[]`

21.22 callback

```
method.callback
```

This property does contain the given `callback`.

Use the `setArguments()` method for setting the parameters and the callback method with the given `IArguments` object.

21.22.1 Returns

`undefined`

21.23 setArguments

```
method.setArguments(arguments)
```

This method will be used to set the given method arguments. The `setArguments` method will set the `parameters` and `callback` property.

21.23.1 Parameters

- `arguments` - `Array`: The `arguments` of the function call.

21.23.2 Returns

`Object`

21.24 getArguments

```
method.getArguments()
```

This method will be used to get the method arguments. The `getArguments` method will return a object with the properties `parameters` and `callback`.

21.24.1 Returns

`Object`

21.25 isHash

```
method.isHash(value)
```

This method will check if the given value is a string and starts with `0x`. It will be used in several methods for deciding which JSON-RPC method should get executed.

21.25.1 Parameters

- `value - string`

21.25.2 Returns

`boolean`

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 22

Core Subscriptions Module

The `Core Subscriptions Module` does provide all the subscriptions classes to extend and execute them.

22.1 AbstractSubscription

Source: AbstractSubscription

The `AbstractSubscription` class extends from the `EventEmitter` object and does have the following constructor parameters:

- *type* - `String` The subscriptions type `eth_subscribe` or `shh_subscribe`.
- *method* - `String` The subscription method which is the first parameter in the JSON-RPC payload object.
- *options* - `Object` The options object of the subscription.
- *formatters* - `Object` The formatters object.
- *moduleInstance* - `AbstractWeb3Module` An `AbstractWeb3Module` instance.

The `AbstractSubscription` class is the base subscription class of all subscriptions.

You're able to overwrite these methods:

- *subscribe*
- *unsubscribe*
- beforeSubscription
- onNewSubscriptionItem

22.2 subscribe

```
subscription.subscribe(callback)
```

This method will start the subscription.

22.2.1 Parameters

- `callback` - `Function`

22.2.2 Returns

`AbstractSubscription`

22.3 unsubscribe

```
subscription.unsubscribe(callback)
```

This method will end the subscription.

22.3.1 Parameters

- `callback` - `Function`

22.3.2 Returns

`Promise<boolean|Error>`

22.4 beforeSubscription

```
subscription.beforeSubscription(moduleInstance)
```

This method will be executed before the subscription happens. The `beforeSubscription` method gives you the possibility to customize the subscription class before the request will be sent.

22.4.1 Parameters

- `moduleInstance` - `AbstractWeb3Module` The current `AbstractWeb3Module`.

22.5 onNewSubscriptionItem

```
subscription.onNewSubscriptionItem(moduleInstance)
```

This method will be executed on each subscription item. The `onNewSubscriptionItem` method gives you the possibility to map the response.

22.5.1 Parameters

- `item` - `any`

22.5.2 Returns

`any`

22.6 type

```
subscription.type
```

The property `type` does contain the subscription type.

22.6.1 Returns

`String` - `eth_subscribe` or `shh_subscribe`

22.7 method

```
subscription.method
```

The property `method` does contain the subscription method.

22.7.1 Returns

`String`

22.8 options

```
subscription.options
```

The property `options` does contain the subscription options.

22.8.1 Returns

`Object`

22.9 id

```
subscription.id
```

The property `id` does contain the subscription id when the subscription is running.

22.9.1 Returns

`String`

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 23

Admin Module

The `web3-eth-admin` package allows you to interact with the Ethereum node's admin management.

```
import Web3 from 'web3';
import {Admin} from 'web3-eth-admin';

// "Web3.givenProvider" will be set if in an Ethereum supported browser.
const admin = new Admin(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);
```

23.1 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

23.1.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

23.1.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

23.2 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

23.2.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block
- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

23.3 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

23.3.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

23.4 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

23.4.1 Returns

`string|number`: The current value of the defaultGasPrice property.

23.5 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

23.5.1 Returns

`string|number`: The current value of the defaultGas property.

23.6 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

23.6.1 Returns

`number`: The current value of transactionBlockTimeout

23.7 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

23.7.1 Returns

`number`: The current value of transactionConfirmationBlocks

23.8 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

23.8.1 Returns

`number`: The current value of transactionPollingTimeout

23.9 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

23.9.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

23.10 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

23.10.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

23.10.2 Returns

`Boolean`

23.10.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

23.11 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

23.11.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

23.11.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));
```

(continues on next page)

(continued from previous page)

```
// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

23.12 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

23.12.1 Returns

`Object`: The given provider set or `false`.

23.12.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

23.13 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

23.13.1 Returns

`Object`: The current provider set.

23.13.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

23.14 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

23.14.1 Parameters

none

23.14.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

23.14.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

23.15 addPeer

```
admin.addPeer(url, [callback])
```

Add an admin peer on the node that Web3 is connected to with its provider. The RPC method used is `admin_addPeer`.

23.15.1 Parameters

1. `url` - `String`: The enode URL of the remote peer.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.15.2 Returns

`Promise<boolean>` - True if peer added successfully.

23.15.3 Example

```
admin.addPeer("enode://
↪a979fb575495b8d6db44f750317d0f4622bf4c2aa3365d6af7c284339968eef29b69ad0dce72a4d8db5ebb4968de0e3bec9
↪16.188.185:30303")
.then(console.log);
> true
```

23.16 getDataDirectory

```
admin.getDataDirectory([, callback])
```

Provides absolute path of the running node, which is used by the node to store all its databases. The RPC method used is `admin_datadir`.

23.16.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.16.2 Returns

`Promise<string>` - The path.

23.16.3 Example

```
admin.getDataDirectory()
.then(console.log);
> "/home/ubuntu/.ethereum"
```

23.17 getNodeInfo

```
admin.getNodeInfo([, callback])
```

This property can be queried for all the information known about the running node at the networking granularity. The RPC method used is `admin_nodeInfo`.

23.17.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.17.2 Returns

`Promise<object>` - The node information array.

- `enode` - `string`: Enode address of the node.
- `id` - `string`: Node Id.
- `listenAddr` - `string`: lister host and port address.
- `name` - `string`: Name of the node, including client type, version, OS, custom data
- `discovery` - `number`: UDP listening port for discovery protocol
- `listener` - `number`: TCP listening port for RLPx
- `difficulty` - `number`: Difficulty level applied during the nonce discovering of this block.
- `genesis` - `string`: Very first block hash.
- `head` - `string`: Current block hash.
- `network` - `number`: currently used Ethereum networks ids.

23.17.3 Example

```
admin.getNodeInfo().then(console.log);
> {
    enode: "enode://
↪44826a5d6a55f88a18298bca4773fca5749cdc3a5c9f308aa7d810e9b31123f3e7c5fba0b1d70aac5308426f47df2a128a
↪",
    id:
↪"44826a5d6a55f88a18298bca4773fca5749cdc3a5c9f308aa7d810e9b31123f3e7c5fba0b1d70aac5308426f47df2a128a
↪",
    ip: "::",
    listenAddr: "[::]:30303",
    name: "Geth/v1.5.0-unstable/linux/go1.6",
    ports: {
        discovery: 30303,
        listener: 30303
    },
    protocols: {
        eth: {
        difficulty: 17334254859343145000,
        genesis: "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
        head: "0xb83f73fbe6220c111136aefd27b160bf4a34085c65ba89f24246b3162257c36a",
        network: 1
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

23.18 getPeers

```
admin.getPeers([, callback])
```

This will provide all the information known about the connected remote nodes at the networking granularity. The RPC method used is `admin_peers`.

23.18.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.18.2 Returns

`Promise<Object>` - List of all connected peers.

- `caps` - `Array`: Protocols advertised by this peer.
- `id` - `string`: Peer node Id.
- `name` - `string`: Peer name of the node, including client type, version, OS, custom data
- `localAddress` - `string`: Local endpoint of the TCP data connection.
- `remoteAddress` - `string`: Remote endpoint of the TCP data connection.
- `difficulty` - `number`: Difficulty level applied during the nonce discovering of this block.
- `head` - `string`: Peer's current block hash.
- `version` - `number`: Version number of the protocol.

23.18.3 Example

```
admin.getPeers().then(console.log);
> [{
        caps: ["eth/61", "eth/62", "eth/63"],
        id:
↪"08a6b39263470c78d3e4f58e3c997cd2e7af623afce64656cfc56480babcea7a9138f3d09d7b9879344c2d2e457679e365
↪",
        name: "Geth/v1.5.0-unstable/linux/go1.5.1",
        network: {
            localAddress: "192.168.0.104:51068",
            remoteAddress: "71.62.31.72:30303"
        },
        protocols: {
            eth: {
                difficulty: 17334052235346465000,
                head:
↪"5794b768dae6c6ee5366e6ca7662bdff2882576e09609bf778633e470e0e7852",
```

(continues on next page)

(continued from previous page)

```
                version: 63
            }
        }
    }, /* ... */ {
        caps: ["eth/61", "eth/62", "eth/63"],
        id:
↪"fcad9f6d3faf89a0908a11ddae9d4be3a1039108263b06c96171eb3b0f3ba85a7095a03bb65198c35a04829032d198759e
↪",
        name: "Geth/v1.3.5-506c9277/linux/go1.4.2",
        network: {
            localAddress: "192.168.0.104:55968",
            remoteAddress: "121.196.232.205:30303"
        },
        protocols: {
        eth: {
            difficulty: 17335165914080772000,
            head: "5794b768dae6c6ee5366e6ca7662bdff2882576e09609bf778633e470e0e7852",
            version: 63
        }
    }
}]
```

23.19 setSolc

```
admin.setSolc(string, [, callback])
```

Sets the Solidity compiler path to be used by the node when invoking the eth_compileSolidity RPC method The RPC method used is `admin_setSolc`.

23.19.1 Parameters

1. `String` - The path of the solidity compiler.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.19.2 Returns

`Promise<string>` - A message.

23.19.3 Example

```
admin.setSolc("/usr/bin/solc").then(console.log);
> "solc, the solidity compiler commandline interface\nVersion: 0.3.2-0/Release-Linux/
↪g++/Interpreter\n\npath: /usr/bin/solc"
```

23.20 startRPC

```
admin.startRPC(host, port, cors, apis [, callback])
```

It starts an HTTP based JSON RPC API webserver to handle client requests. All the parameters are optional. The RPC method used is `admin_startRPC`.

23.20.1 Parameters

1. `host` - `String` - (optional) The network interface to open the listener socket on (defaults to "localhost").
2. `port` - `number` - (optional) The network port to open the listener socket on (defaults to 8545).
3. `cors` - `string` - (optional) Cross-origin resource sharing header to use (defaults to "").
4. `apis` - `string` - (optional) API modules to offer over this interface (defaults to "eth,net,web3").
5. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.20.2 Returns

`Promise<boolean>` - True if Remote Procedure Call (RPC) got started.

23.20.3 Example

```
admin.startRPC("127.0.0.1", 8545)
.then(console.log('RPC Started!'));
> "RPC Started!"
```

23.21 startWS

```
admin.startWS(host, port, cors, apis [, callback])
```

It starts an WebSocket based JSON RPC API webserver to handle client requests. All the parameters are optional. The RPC method used is `admin_startWS`.

23.21.1 Parameters

1. `host` - `String` - (optional) The network interface to open the listener socket on (defaults to "localhost").
2. `port` - `number` - (optional) The network port to open the listener socket on (defaults to 8545).
3. `cors` - `string` - (optional) Cross-origin resource sharing header to use (defaults to "").
4. `apis` - `string` - (optional) API modules to offer over this interface (defaults to "eth,net,web3").
5. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.21.2 Returns

`Promise<boolean>` - True if Web socket (WS) got started.

23.21.3 Example

```
admin.startRPC("127.0.0.1", 8546)
.then(console.log('WS Started!'));
> "WS Started!"
```

23.22 stopRPC

```
admin.stopRPC([, callback])
```

This method closes the currently open HTTP RPC endpoint. As the node can only have a single HTTP endpoint running, this method takes no parameters, returning a boolean whether the endpoint was closed or not. The RPC method used is `admin_stopRPC`.

23.22.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.22.2 Returns

`Promise<boolean>` - True if Remote Procedure Call (RPC) successfully stopped.

23.22.3 Example

```
admin.stopRPC().then(console.log);
> true
```

23.23 stopWS

```
admin.stopWS([, callback])
```

This method closes the currently open WebSocket RPC endpoint. As the node can only have a single WebSocket endpoint running, this method takes no parameters, returning a boolean whether the endpoint was closed or not. The RPC method used is `admin_stopWS`.

23.23.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

23.23.2 Returns

`Promise<boolean>` - True if Web Socket (WS) successfully stopped.

23.23.3 Example

```
admin.stopWS().then(console.log);
> true
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 24

Debug Module

The `web3-eth-debug` module allows you to interact with the Ethereum node's debug methods.

```
import Web3 from 'web3';
import {Debug} from 'web3-eth-debug';

// "Web3.givenProvider" will be set if in an Ethereum supported browser.
const debug = new Debug(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);
```

24.1 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

24.1.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

24.1.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

24.2 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

24.2.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block
- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

24.3 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

24.3.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

24.4 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

24.4.1 Returns

`string|number`: The current value of the defaultGasPrice property.

24.5 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

24.5.1 Returns

`string|number`: The current value of the defaultGas property.

24.6 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

24.6.1 Returns

`number`: The current value of transactionBlockTimeout

24.7 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

24.7.1 Returns

`number`: The current value of transactionConfirmationBlocks

24.8 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

24.8.1 Returns

`number`: The current value of transactionPollingTimeout

24.9 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

24.9.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

24.10 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

24.10.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

24.10.2 Returns

`Boolean`

24.10.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

24.11 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

24.11.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

24.11.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));
```

(continues on next page)

(continued from previous page)

```
// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

24.12 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

24.12.1 Returns

`Object`: The given provider set or `false`.

24.12.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

24.13 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

24.13.1 Returns

`Object`: The current provider set.

24.13.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

24.14 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

24.14.1 Parameters

none

24.14.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

24.14.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

24.15 setBackTraceAt

```
debug.setBackTraceAt(location, [callback])
```

Sets the logging backtrace location.

24.15.1 Parameters

1. `location` - `String`: The location is specified as `<filename>:<line>`.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.15.2 Returns

`Promise<null>`

24.15.3 Example

```
admin.setBackTraceAt('filename.go:200').then(console.log);
```

24.16 blockProfile

```
debug.blockProfile(file, seconds, [, callback])
```

Turns on block profiling for the given duration and writes profile data to disk. If a custom rate is desired, set the rate and write the profile manually using `debug.writeBlockProfile`.

24.16.1 Parameters

1. `file` - `String` 1. `seconds` - `Number|String` The seconds as Hex string or number. 2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.16.2 Returns

`Promise<null>`

24.16.3 Example

```
debug.blockProfile('file', 100).then(console.log);
> null
```

24.17 cpuProfile

```
debug.cpuProfile(file, seconds, [, callback])
```

Turns on CPU profiling for the given duration and writes profile data to disk.

24.17.1 Parameters

1. `file` - `String` 1. `seconds` - `Number | String` The seconds as Hex string or number. 2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.17.2 Returns

`Promise<null>`

24.17.3 Example

```
debug.cpuProfile('file', 100).then(console.log);
> null
```

24.18 dumpBlock

```
debug.dumpBlock(blockNumber, [, callback])
```

Retrieves the state that corresponds to the block number and returns a list of accounts (including storage and code).

24.18.1 Parameters

1. `blockNumber` - `Number | String` The block number as Hex string or number.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.18.2 Returns

`Promise<Object>`

24.18.3 Example

```
debug.dumpBlock('file', 100).then(console.log);
{
    root: "19f4ed94e188dd9c7eb04226bd240fa6b449401a6c656d6d2816a87ccaf206f1",
    accounts: {
        fff7ac99c8e4feb60c9750054bdc14ce1857f181: {
            balance: "49358640978154672",
            code: "",
            codeHash:
↪"c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470",
            nonce: 2,
            root: "56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
            storage: {}
        },
        fffbca3a38c3c5fcb3adbb8e63c04c3e629aafce: {
```

(continues on next page)

(continued from previous page)

```
            balance: "3460945928",
            code: "",
            codeHash:
↪"c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470",
            nonce: 657,
            root: "56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
            storage: {}
        }
    },
}
```

24.19 getGCStats

```
debug.getGCStats([, callback])
```

Returns GC statistics. See https://golang.org/pkg/runtime/debug/#GCStats for information about the fields of the returned object.

24.19.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.19.2 Returns

`Promise<Object>`

24.19.3 Example

```
debug.getGCStats().then(console.log);
```

24.20 getBlockRlp

```
debug.getBlockRlp(number, [, callback])
```

Retrieves and returns the RLP encoded block by number.

24.20.1 Parameters

1. `number` - `Number | String` The block number as hex string or number.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.20.2 Returns

`Promise<string>`

24.20.3 Example

```
debug.getBlockRlp(100).then(console.log);
> '0x0'
```

24.21 goTrace

```
debug.goTrace(file, seconds, [, callback])
```

Turns on Go runtime tracing for the given duration and writes trace data to disk.

24.21.1 Parameters

1. `file` - `String` 1. `seconds` - `Number | String` The seconds as Hex string or number. 2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.21.2 Returns

`Promise<null>`

24.21.3 Example

```
debug.goTrace('file', 100).then(console.log);
> null
```

24.22 getMemStats

```
debug.getMemStats([, callback])
```

Returns detailed runtime memory statistics.

24.22.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.22.2 Returns

`Promise<Object>`

24.22.3 Example

```
debug.getMemStats().then(console.log);
> MemStats // MemStats object from Go
```

24.23 getSeedHash

```
debug.getSeedHash(number, [, callback])
```

Fetches and retrieves the seed hash of the block by number

24.23.1 Parameters

1. `number` - `Number | String` The block number as Hex string or number. 1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.23.2 Returns

`Promise<string>`

24.23.3 Example

```
debug.getSeedHash().then(console.log);
> '0x0'
```

24.24 setBlockProfileRate

```
debug.setBlockProfileRate(rate, [, callback])
```

Sets the rate (in samples/sec) of goroutine block profile data collection. A non-zero rate enables block profiling, setting it to zero stops the profile. Collected profile data can be written using `debug.writeBlockProfile`.

24.24.1 Parameters

1. `number` - `Number | String` The block profile rate as number or Hex string.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.24.2 Returns

`Promise<null>`

24.24.3 Example

```
debug.setBlockProfileRate().then(console.log);
> null
```

24.25 setHead

```
debug.setHead(number, [, callback])
```

Sets the current head of the local chain by block number. Note, this is a destructive action and may severely damage your chain. Use with extreme caution.

24.25.1 Parameters

1. `number` - `Number | String` The block number as Hex string or number.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.25.2 Returns

`Promise<null>`

24.25.3 Example

```
debug.setHead(100).then(console.log);
> null
```

24.26 getStacks

```
debug.getStacks([, callback])
```

Returns a printed representation of the stacks of all goroutines.

24.26.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.26.2 Returns

`Promise<string>`

24.26.3 Example

```
debug.getStacks().then(console.log);
```

24.27 startCPUProfile

```
debug.startCPUProfile(file, [, callback])
```

Turns on CPU profiling indefinitely, writing to the given file.

24.27.1 Parameters

1. `file` - `String`
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.27.2 Returns

`Promise<null>`

24.27.3 Example

```
debug.startCPUProfile().then(console.log);
> null
```

24.28 stopCPUProfile

```
debug.stopCPUProfile([, callback])
```

Stops an ongoing CPU profile.

24.28.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.28.2 Returns

`Promise<null>`

24.28.3 Example

```
debug.stopCPUProfile().then(console.log);
> null
```

24.29 startGoTrace

```
debug.startGoTrace(file, [, callback])
```

Turns on CPU profiling indefinitely, writing to the given file.

24.29.1 Parameters

1. `file` - `String`
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.29.2 Returns

`Promise<null>`

24.29.3 Example

```
debug.startGoTrace('file').then(console.log);
> null
```

24.30 stopGoTrace

```
debug.stopGoTrace([, callback])
```

Stops writing the Go runtime trace.

24.30.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.30.2 Returns

`Promise<null>`

24.30.3 Example

```
debug.stopGoTrace().then(console.log);
> null
```

24.31 getBlockTrace

```
debug.getBlockTrace(blockRlp, options, [, callback])
```

The traceBlock method will return a full stack trace of all invoked opcodes of all transaction that were included included in this block. Note, the parent of this block must be present or it will fail.

24.31.1 Parameters

1. `blockRlp` - `String` RLP encoded block
2. `options` - `Object` The block trace object
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.31.2 Returns

`Promise<Object>`

24.31.3 Example

```
debug.getBlockTrace('0x0', {}).then(console.log);
> {
    gas: 85301,
    returnValue: "",
    structLogs: [{...}]
}
```

24.32 getBlockTraceByNumber

```
debug.getBlockTraceByNumber(number, options, [, callback])
```

The traceBlockByNumber method accepts a block number and will replay the block that is already present in the database.

24.32.1 Parameters

1. `number` - `Number | String` The block number as Hex string or number.
2. `options` - `Object` The block trace object
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.32.2 Returns

`Promise<Object>`

24.32.3 Example

```
debug.getBlockTraceByNumber(100, {}).then(console.log);
> {
    gas: 85301,
    returnValue: "",
    structLogs: [{...}]
}
```

24.33 getBlockTraceByHash

```
debug.getBlockTraceByHash(hash, options, [, callback])
```

The traceBlockByHash accepts a block hash and will replay the block that is already present in the database.

24.33.1 Parameters

1. `hash` - `String` The block hash
2. `options` - `Object` The block trace object
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.33.2 Returns

`Promise<Object>`

24.33.3 Example

```
debug.getBlockTraceByHash('0x0', {}).then(console.log);
> {
    gas: 85301,
    returnValue: "",
    structLogs: [{...}]
}
```

24.34 getBlockTraceFromFile

```
debug.getBlockTraceFromFile(fileName, options, [, callback])
```

The traceBlockFromFile accepts a file containing the RLP of the block.

24.34.1 Parameters

1. `fileName` - `String` The file name
2. `options` - `Object` The block trace object
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.34.2 Returns

`Promise<Object>`

24.34.3 Example

```
debug.getBlockTraceFromFile('filename', {}).then(console.log);
> {
    gas: 85301,
    returnValue: "",
    structLogs: [{...}]
}
```

24.35 getTransactionTrace

```
debug.getTransactionTrace(txHash, options, [, callback])
```

The traceTransaction debugging method will attempt to run the transaction in the exact same manner as it was executed on the network. It will replay any transaction that may have been executed prior to this one before it will finally attempt to execute the transaction that corresponds to the given hash.

In addition to the hash of the transaction you may give it a secondary optional argument, which specifies the options for this specific call.

The possible options are:

1. `disableStorage` - `boolean` Setting this to true will disable storage capture (default = false). 1. `disableMemory` - `boolean` Setting this to true will disable memory capture (default = false). 1. `disableStack` - `boolean` Setting this to true will disable stack capture (default = false). 1. `tracer` - `string` Setting this will enable JavaScript-based transaction tracing, described below. If set, the previous four arguments will be ignored. 1. `timeout` - `string` Overrides the default timeout of 5 seconds for JavaScript-based tracing calls

JSON-RPC specification for debug_traceTransaction

24.35.1 Parameters

1. `txHash` - `String` The transaction hash
2. `options` - `Object` The block trace object
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.35.2 Returns

`Promise<Object>`

24.35.3 Example

```
debug.getTransactionTrace('0x0', {}).then(console.log);
> {
    gas: 85301,
    returnValue: "",
    structLogs: [{...}]
}
```

24.36 setVerbosity

```
debug.setVerbosity(level, [, callback])
```

Sets the logging verbosity ceiling. Log messages with level up to and including the given level will be printed. The verbosity of individual packages and source files can be raised using `debug.setVerbosityPattern`.

24.36.1 Parameters

1. `level` - `Number | String` The verbosity level as Hex string or number. 1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.36.2 Returns

`Promise<null>`

24.36.3 Example

```
debug.setVerbosity(1).then(console.log);
> null
```

24.37 setVerbosityPattern

```
debug.setVerbosityPattern(pattern, [, callback])
```

Sets the logging verbosity pattern.

24.37.1 Parameters

1. `pattern` - `String` The verbosity pattern 1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.37.2 Returns

`Promise<null>`

24.37.3 Example

```
// If you want to see messages from a particular Go package (directory) and all␣
↪subdirectories, use:
debug.setVerbosityPattern('eth/*=6').then(console.log);
> null

// If you want to restrict messages to a particular package (e.g. p2p) but exclude␣
↪subdirectories, use:
debug.setVerbosityPattern('p2p=6').then(console.log);
> null

// If you want to see log messages from a particular source file, use:
debug.setVerbosityPattern('server.go=6').then(console.log);
> null

// You can compose these basic patterns. If you want to see all output from peer.go␣
↪in a package below eth
// (eth/peer.go, eth/downloader/peer.go) as well as output from package p2p at level
↪<= 5, use:
debug.setVerbosityPattern('eth/*/peer.go=6,p2p=5').then(console.log);
> null
```

24.38 writeBlockProfile

```
debug.writeBlockProfile(file, [, callback])
```

Writes a goroutine blocking profile to the given file.

24.38.1 Parameters

1. `file` - `String` The file 1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.38.2 Returns

`Promise<null>`

24.38.3 Example

```
debug.writeBlockProfile('file').then(console.log);
> null
```

24.39 writeMemProfile

```
debug.writeMemProfile(file, [, callback])
```

Writes an allocation profile to the given file.

24.39.1 Parameters

1. `file` - `String` The file 1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

24.39.2 Returns

`Promise<null>`

24.39.3 Example

```
debug.writeBlockProfile('file').then(console.log);
> null
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 25

Miner Module

The `web3-eth-miner` package allows you to remote control the node's mining operation and set various mining specific settings.

```
import {Miner} from 'web3-eth-miner';

// "Web3.givenProvider" will be set if in an Ethereum supported browser.
const miner = new Miner(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',␣
↪null, options);
```

25.1 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

25.1.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

25.1.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

25.2 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

25.2.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block
- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

25.3 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

25.3.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

25.4 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

25.4.1 Returns

`string|number`: The current value of the defaultGasPrice property.

25.5 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

25.5.1 Returns

`string|number`: The current value of the defaultGas property.

25.6 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

25.6.1 Returns

`number`: The current value of transactionBlockTimeout

25.7 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

25.7.1 Returns

`number`: The current value of transactionConfirmationBlocks

25.8 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

25.8.1 Returns

`number`: The current value of transactionPollingTimeout

25.9 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

25.9.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

25.10 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

25.10.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

25.10.2 Returns

`Boolean`

25.10.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

25.11 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

25.11.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

25.11.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));
```

(continues on next page)

(continued from previous page)

```
// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

25.12 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

25.12.1 Returns

`Object`: The given provider set or `false`.

25.12.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

25.13 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

25.13.1 Returns

`Object`: The current provider set.

25.13.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

25.14 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

25.14.1 Parameters

none

25.14.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

25.14.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

25.15 setExtra

```
miner.setExtra(extraData, [, callback])
```

This method allows miner to set extra data during mining the block. The RPC method used is `miner_setExtra`.

25.15.1 Parameters

1. `extraData` - `String`: Extra data which is to be set.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

25.15.2 Returns

`Promise<boolean>` - True if successful.

25.15.3 Example

```
miner.setExtra('Hello').then(console.log);
> true
```

25.16 setGasPrice

```
miner.setGasPrice(gasPrice, [, callback])
```

This method allows to set minimal accepted gas price during mining transactions. Any transactions that are below this limit will get excluded from the mining process. The RPC method used is `miner_setGasPrice`. ———- Parameters ———-

1. `Number | String` - Gas price.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

25.16.1 Returns

`Promise<boolean>` - True if successful.

25.16.2 Example

```
miner.setGasPrice("0x4a817c800").then(console.log);
> true

miner.setGasPrice(20000000000).then(console.log);
> true
```

25.17 setEtherBase

```
miner.setEtherBase(address, [, callback])
```

Sets etherbase, where mining reward will go. The RPC method used is `miner_setEtherbase`.

25.17.1 Parameters

1. `String` - address where mining reward will go.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

25.17.2 Returns

`Promise<boolean>` - True if successful.

25.17.3 Example

```
miner.setEtherBase("0x3d80b31a78c30fc628f20b2c89d7ddbf6e53cedc").then(console.log);
> true
```

25.18 startMining

```
miner.startMining(miningThread, [, callback])
```

Start the CPU mining process with the given number of threads. The RPC method used is `miner_start`.

25.18.1 Parameters

1. `Number | String` - Mining threads.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

25.18.2 Returns

`Promise<boolean>` - True if successful.

25.18.3 Example

```
miner.startMining('0x1').then(console.log);
> true

miner.startMining(1).then(console.log);
> true
```

25.19 stopMining

```
miner.stopMining([callback])
```

Stop the CPU mining process. The RPC method used is `miner_stop`.

25.19.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

25.19.2 Returns

`Promise<boolean>` - True if successful.

25.19.3 Example

```
miner.stopMining().then(console.log);
> true
```

Note: This documentation is under construction and documents the 2.x alpha versions of web3.js. The current stable version of web3.js is 1.0 and should get preferred for production use cases.

CHAPTER 26

TxPool Module

The `web3-eth-txpool` package gives you access to several non-standard RPC methods to inspect the contents of the transaction pool containing all the currently pending transactions as well as the ones queued for future processing.

```
import Web3 from 'web3';
import {TxPool} from 'web3-eth-txpool';

// "Web3.givenProvider" will be set if in an Ethereum supported browser.
const txPool = new TxPool(Web3.givenProvider || 'ws://some.local-or-remote.node:8546',
↪ null, options);
```

26.1 options

An Web3 module does provide several options for configuring the transaction confirmation worklfow or for defining default values. These are the currently available option properties on a Web3 module:

26.1.1 Module Options

defaultAccount

defaultBlock

defaultGas

defaultGasPrice

transactionBlockTimeout

transactionConfirmationBlocks

transactionPollingTimeout

transactionSigner

26.1.2 Example

```
import Web3 from 'web3';

const options = {
    defaultAccount: '0x0',
    defaultBlock: 'latest',
    defaultGas: 1,
    defaultGasPrice: 0,
    transactionBlockTimeout: 50,
    transactionConfirmationBlocks: 24,
    transactionPollingTimeout: 480,
    transactionSigner: new CustomTransactionSigner()
}

const web3 = new Web3('http://localhost:8545', null, options);
```

26.2 defaultBlock

```
web3.defaultBlock
web3.eth.defaultBlock
web3.shh.defaultBlock
...
```

The default block is used for all methods which have a block parameter. You can override it by passing the block parameter if a block is required.

Example:

- *web3.eth.getBalance()*
- *web3.eth.getCode()*
- *web3.eth.getTransactionCount()*
- *web3.eth.getStorageAt()*
- *web3.eth.call()*
- *new web3.eth.Contract() -> myContract.methods.myMethod().call()*

26.2.1 Returns

The `defaultBlock` property can return the following values:

- `Number`: A block number
- `"genesis"` - `String`: The genesis block
- `"latest"` - `String`: The latest block (current head of the blockchain)
- `"pending"` - `String`: The currently mined block (including pending transactions)

Default is `"latest"`

26.3 defaultAccount

```
web3.defaultAccount
web3.eth.defaultAccount
web3.shh.defaultAccount
...
```

This default address is used as the default `"from"` property, if no `"from"` property is specified.

26.3.1 Returns

`String` - 20 Bytes: Any Ethereum address. You need to have the private key for that address in your node or keystore. (Default is `undefined`)

26.4 defaultGasPrice

```
web3.defaultGasPrice
web3.eth.defaultGasPrice
web3.shh.defaultGasPrice
...
```

The default gas price which will be used for a request.

26.4.1 Returns

`string|number`: The current value of the defaultGasPrice property.

26.5 defaultGas

```
web3.defaultGas
web3.eth.defaultGas
web3.shh.defaultGas
...
```

The default gas which will be used for a request.

26.5.1 Returns

`string|number`: The current value of the defaultGas property.

26.6 transactionBlockTimeout

```
web3.transactionBlockTimeout
web3.eth.transactionBlockTimeout
web3.shh.transactionBlockTimeout
...
```

The `transactionBlockTimeout` will be used over a socket based connection. This option does define the amount of new blocks it should wait until the first confirmation happens. This means the PromiEvent rejects with a timeout error when the timeout got exceeded.

26.6.1 Returns

`number`: The current value of transactionBlockTimeout

26.7 transactionConfirmationBlocks

```
web3.transactionConfirmationBlocks
web3.eth.transactionConfirmationBlocks
web3.shh.transactionConfirmationBlocks
...
```

This defines the number of blocks it requires until a transaction will be handled as confirmed.

26.7.1 Returns

`number`: The current value of transactionConfirmationBlocks

26.8 transactionPollingTimeout

```
web3.transactionPollingTimeout
web3.eth.transactionPollingTimeout
web3.shh.transactionPollingTimeout
...
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option does define the amount of polls (each second) it should wait until the first confirmation happens.

26.8.1 Returns

`number`: The current value of transactionPollingTimeout

26.9 transactionSigner

```
web3.eth.transactionSigner
...
```

The `transactionSigner` property does provide us the possibility to customize the signing process of the `Eth` module and the related sub-modules.

The interface of a `TransactionSigner`:

```
interface TransactionSigner {
    sign(txObject: Transaction): Promise<SignedTransaction>
}

interface SignedTransaction {
    messageHash: string,
    v: string,
    r: string,
    s: string,
    rawTransaction: string
}
```

26.9.1 Returns

`TransactionSigner`: A JavaScript class of type TransactionSigner.

26.10 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `web3` it will also set the provider for all sub modules `web3.eth`, `web3.shh`, etc.

26.10.1 Parameters

1. `Object|String` - `provider`: a valid provider
2. `Net` - `net`: (optional) the node.js Net package. This is only required for the IPC provider.

26.10.2 Returns

`Boolean`

26.10.3 Example

```
import Web3 from 'web3';

const web3 = new Web3('http://localhost:8545');

// or
const web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

// change provider
web3.setProvider('ws://localhost:8546');
// or
web3.setProvider(new Web3.providers.WebsocketProvider('ws://localhost:8546'));

// Using the IPC provider in node.js
const net = require('net');
const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path

// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

26.11 providers

```
Web3.providers
Eth.providers
...
```

Contains the current available providers.

26.11.1 Value

`Object` with the following providers:

- `Object` - `HttpProvider`: The HTTP provider is **deprecated**, as it won't work for subscriptions.
- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.
- `Object` - `IpcProvider`: The IPC provider is used node.js dapps when running a local node. Gives the most secure connection.

26.11.2 Example

```
const Web3 = require('web3');
// use the given Provider, e.g in Mist, or instantiate a new websocket provider
const web3 = new Web3(Web3.givenProvider || 'ws://localhost:8546');
// or
const web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider('ws:/
↪/localhost:8546'));
```

(continues on next page)

(continued from previous page)

```
// Using the IPC provider in node.js
const net = require('net');

const web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
const web3 = new Web3(new Web3.providers.IpcProvider('/Users/myuser/Library/Ethereum/
↪geth.ipc', net)); // mac os path
// on windows the path is: '\\\\.\\pipe\\geth.ipc'
// on linux the path is: '/users/myuser/.ethereum/geth.ipc'
```

26.12 givenProvider

```
Web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
...
```

When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise `null`.

26.12.1 Returns

`Object`: The given provider set or `false`.

26.12.2 Example

```
web3.setProvider(Web3.givenProvider || 'ws://localhost:8546');
```

26.13 currentProvider

```
web3.currentProvider
web3.eth.currentProvider
web3.shh.currentProvider
...
```

Will return the current provider.

26.13.1 Returns

`Object`: The current provider set.

26.13.2 Example

```
if (!web3.currentProvider) {
    web3.setProvider('http://localhost:8545');
}
```

26.14 BatchRequest

```
new web3.BatchRequest()
new web3.eth.BatchRequest()
new web3.shh.BatchRequest()
...
```

Class to create and execute batch requests.

26.14.1 Parameters

none

26.14.2 Returns

`Object`: With the following methods:

- `add(request)`: To add a request object to the batch call.
- `execute()`: Will execute the batch request.

26.14.3 Example

```
const contract = new web3.eth.Contract(abi, address);

const batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest'));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}));
batch.execute().then(...);
```

26.15 getContent

```
txPool.getContent([callback])
```

This API can be used to list the exact details of all the transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future executions. The RPC method used is `txpool_content`.

26.15.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

26.15.2 Returns

`Promise<Object>` - The list of pending as well as queued transactions.

- `pending` - `Object`: List of pending transactions with transaction details.
- `queued` - `Object`: List of queued transactions with transaction details.
  - `hash` 32 Bytes - `String`: Hash of the transaction.
  - `nonce` - `Number`: The number of transactions made by the sender prior to this one.
  - `blockHash` 32 Bytes - `String`: Hash of the block where this transaction was in. `null` when its pending.
  - `blockNumber` - `Number`: Block number where this transaction was in. `null` when its pending.
  - `transactionIndex` - `Number`: Integer of the transactions index position in the block. `null` when its pending.
  - `from` - `String`: Address of the sender.
  - `to` - `String`: Address of the receiver. `null` when its a contract creation transaction.
  - `value` - `String`: Value transferred in wei.
  - `gasPrice` - `String`: The wei per unit of gas provided by the sender in wei.
  - `gas` - `Number`: Gas provided by the sender.
  - `input` - `String`: The data sent along with the transaction.

26.15.3 Example

```
txPool.getContent().then(console.log);
> {
    pending: {
        0x0216d5032f356960cd3749c31ab34eeff21b3395: {
        806: [{
            blockHash:
↪"0x0000000000000000000000000000000000000000000000000000000000000000",
            blockNumber: null,
            from: "0x0216d5032f356960cd3749c31ab34eeff21b3395",
            gas: "0x5208",
            gasPrice: "0xba43b7400",
            hash: "0xaf953a2d01f55cfe080c0c94150a60105e8ac3d51153058a1f03dd239dd08586
↪",
            input: "0x",
            nonce: "0x326",
            to: "0x7f69a91a3cf4be60020fb58b893b7cbb65376db8",
            transactionIndex: null,
            value: "0x19a99f0cf456000"
        }]
        },
        0x24d407e5a0b506e1cb2fae163100b5de01f5193c: {
```

(continues on next page)

(continued from previous page)

```
        34: [{
            blockHash:
↪"0x0000000000000000000000000000000000000000000000000000000000000000",
            blockNumber: null,
            from: "0x24d407e5a0b506e1cb2fae163100b5de01f5193c",
            gas: "0x44c72",
            gasPrice: "0x4a817c800",
            hash: "0xb5b8b853af32226755a65ba0602f7ed0e8be2211516153b75e9ed640a7d359fe
↪",
            input:
↪"0xb61d27f6000000000000000000000000024d407e5a0b506e1cb2fae163100b5de01f5193c000000000000000000000000
↪",
            nonce: "0x22",
            to: "0x7320785200f74861b69c49e4ab32399a71b34f1a",
            transactionIndex: null,
            value: "0x0"
        }]
        }
    },
    queued: {
        0x976a3fc5d6f7d259ebfb4cc2ae75115475e9867c: {
            3: [{
                blockHash:
↪"0x0000000000000000000000000000000000000000000000000000000000000000",
                blockNumber: null,
                from: "0x976a3fc5d6f7d259ebfb4cc2ae75115475e9867c",
                gas: "0x15f90",
                gasPrice: "0x4a817c800",
                hash:
↪"0x57b30c59fc39a50e1cba90e3099286dfa5aaf60294a629240b5bbec6e2e66576",
                input: "0x",
                nonce: "0x3",
                to: "0x346fb27de7e7370008f5da379f74dd49f5f2f80f",
                transactionIndex: null,
                value: "0x1f161421c8e0000"
            }]
        },
        0x9b11bf0459b0c4b2f87f8cebca4cfc26f294b63a: {
            2: [{
                blockHash:
↪"0x0000000000000000000000000000000000000000000000000000000000000000",
                blockNumber: null,
                from: "0x9b11bf0459b0c4b2f87f8cebca4cfc26f294b63a",
                gas: "0x15f90",
                gasPrice: "0xba43b7400",
                hash:
↪"0x3a3c0698552eec2455ed3190eac3996feccc806970a4a056106deaf6ceb1e5e3",
                input: "0x",
                nonce: "0x2",
                to: "0x24a461f25ee6a318bdef7f33de634a67bb67ac9d",
                transactionIndex: null,
                value: "0xebec21ee1da40000"
            }]
        }
    }
}
```

26.16 getInspection

```
txPool.getInspection([, callback])
```

The property can be queried to list a textual summary of all the transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future executions. The RPC method used is `txpool_inspect`.

26.16.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

26.16.2 Returns

`Promise<Object>` - The List of pending and queued transactions summary.

- `pending` - `Object`: List of pending transactions with transaction details.
- `queued` - `Object`: List of queued transactions with transaction details.

26.16.3 Example

```
txPool.getInspection().then(console.log);
> {
    pending: {
        0x26588a9301b0428d95e6fc3a5024fce8bec12d51: {
        31813: ["0x3375ee30428b2a71c428afa5e89e427905f95f7e: 0 wei + 500000 ×␣
↪20000000000 gas"]
        },
        0x2a65aca4d5fc5b5c859090a6c34d164135398226: {
        563662: ["0x958c1fa64b34db746925c6f8a3dd81128e40355e: 1051546810000000000 wei␣
↪+ 90000 × 20000000000 gas"],
        563663: ["0x77517b1491a0299a44d668473411676f94e97e34: 1051190740000000000 wei␣
↪+ 90000 × 20000000000 gas"],
        563664: ["0x3e2a7fe169c8f8eee251bb00d9fb6d304ce07d3a: 1050828950000000000 wei␣
↪+ 90000 × 20000000000 gas"],
        563665: ["0xaf6c4695da477f8c663ea2d8b768ad82cb6a8522: 1050544770000000000 wei␣
↪+ 90000 × 20000000000 gas"],
        563666: ["0x139b148094c50f4d20b01caf21b85edb711574db: 1048598530000000000 wei␣
↪+ 90000 × 20000000000 gas"],
        563667: ["0x48b3bd66770b0d1eecefce090dafee36257538ae: 1048367260000000000 wei␣
↪+ 90000 × 20000000000 gas"],
        563668: ["0x468569500925d53e06dd0993014ad166fd7dd381: 1048126690000000000 wei␣
↪+ 90000 × 20000000000 gas"],
        563669: ["0x3dcb4c90477a4b8ff7190b79b524773cbe3be661: 1047965690000000000 wei␣
↪+ 90000 × 20000000000 gas"],
        563670: ["0x6dfef5bc94b031407ffe71ae8076ca0fbf190963: 1047859050000000000 wei␣
↪+ 90000 × 20000000000 gas"]
        },
        0x9174e688d7de157c5c0583df424eaab2676ac162: {
        3: ["0xbb9bc244d798123fde783fcc1c72d3bb8c189413: 30000000000000000000 wei +␣
↪85000 × 21000000000 gas"]
        },
```

(continues on next page)

(continued from previous page)

```
        0xb18f9d01323e150096650ab989cfecd39d757aec: {
        777: ["0xcd79c72690750f079ae6ab6ccd7e7aedc03c7720: 0 wei + 1000000 ×␣
↪20000000000 gas"]
        },
        0xb2916c870cf66967b6510b76c07e9d13a5d23514: {
        2: ["0x576f25199d60982a8f31a8dff4da8acb982e6aba: 26000000000000000000 wei +␣
↪90000 × 20000000000 gas"]
        },
        0xbc0ca4f217e052753614d6b019948824d0d8688b: {
        0: ["0x2910543af39aba0cd09dbb2d50200b3e800a63d2: 1000000000000000000 wei +␣
↪50000 × 1171602790622 gas"]
        },
        0xea674fdde714fd979de3edf0f56aa9716b898ec8: {
        70148: ["0xe39c55ead9f997f7fa20ebe40fb4649943d7db66: 1000767667434026200 wei␣
↪+ 90000 × 20000000000 gas"]
        }
    },
    queued: {
        0x0f6000de1578619320aba5e392706b131fb1de6f: {
        6: ["0x8383534d0bcd0186d326c993031311c0ac0d9b2d: 9000000000000000000 wei +␣
↪21000 × 20000000000 gas"]
        },
        0x5b30608c678e1ac464a8994c3b33e5cdf3497112: {
        6: ["0x9773547e27f8303c87089dc42d9288aa2b9d8f06: 5000000000000000000 wei +␣
↪90000 × 50000000000 gas"]
        },
        0x976a3fc5d6f7d259ebfb4cc2ae75115475e9867c: {
        3: ["0x346fb27de7e7370008f5da379f74dd49f5f2f80f: 140000000000000000 wei +␣
↪90000 × 20000000000 gas"]
        },
        0x9b11bf0459b0c4b2f87f8cebca4cfc26f294b63a: {
        2: ["0x24a461f25ee6a318bdef7f33de634a67bb67ac9d: 1700000000000000000 wei +␣
↪90000 × 50000000000 gas"],
        6: ["0x6368f3f8c2b42435d6c136757382e4a59436a681: 17990000000000000000 wei +␣
↪90000 × 20000000000 gas", "0x8db7b4e0ecb095fbd01dffa62010801296a9ac78:␣
↪16998950000000000000 wei + 90000 × 20000000000 gas"],
        7: ["0x6368f3f8c2b42435d6c136757382e4a59436a681: 17900000000000000000 wei +␣
↪90000 × 20000000000 gas"]
        }
    }
}
```

26.17 getStatus

```
txPool.getStatus([, callback])
```

This will provide the number of transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future executions. The RPC method used is `txpool_status`.

26.17.1 Parameters

1. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

26.17.2 Returns

`Promise<Object>` - A list of number of pending and queued transactions.

- `pending` - `number`: Number of pending transactions.
- `queued` - `number`: Number of queued transactions.

26.17.3 Example

```
txPool.getStatus().then(console.log);
> {
    pending: 10,
    queued: 7
}
```

Index