web3.js Documentation

Release 1.0.0

Fabian Vogelsteller, Marek Kotewicz, Jeffrey Wilcke, Marian Oance

Apr 25, 2020

User Documentation

web3.js is a collection of libraries which allow you to interact with a local or remote ethereum node, using a HTTP 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

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-bzz` is for the swarm protocol, the decentralized file storage
- 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`
- yarn: `yarn add web3`
- pure js: link the `dist/web3.min.js`

After that you need to create a web3 instance and set a provider. Ethereum supported Browsers like Mist or MetaMask will have a `ethereumProvider` or `web3.currentProvider` available. For web3.js, check `Web3.givenProvider`. If this property is `null` you should connect to a remote/local node.

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

var web3 = new Web3(Web3.givenProvider || "ws://localhost:8545");
```

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

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. This "promiEvent" is a promise combined with an event emitter to allow acting on different stages of action on the blockchain, like a transaction.

PromiEvents work like a normal promises with added `on`, `once` and `off` functions. This way developers can watch for additional events like on "receipt" or "transactionHash".

```
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
});
```

CHAPTER 3

Glossary

3.1 json interface

The json interface is a json object describing the Application Binary Interface (ABI) for an Ethereum smart contract.

Using this json interface web3.js is able to create JavaScript object representing the smart contract and its methods and events using the *web3.eth.Contract object*.

3.1.1 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`.

3.1.2 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
}]
```

CHAPTER 4

Web3

This is the main (or 'umbrella') class of the web3.js library.

```
var Web3 = require('web3');

> Web3.utils
> Web3.version
> Web3.givenProvider
> Web3.providers
> Web3.modules
```

4.1 Web3.modules

```
Web3.modules
```

Will return an object with the classes of all major sub modules, to be able to instantiate them manually.

4.1.1 Returns

`Object`: A list of module constructors:

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

4.1.2 Example

```
Web3.modules
> {
    Eth: Eth(provider),
    Net: Net(provider),
    Personal: Personal(provider),
    Shh: Shh(provider),
    Bzz: Bzz(provider),
}
```

4.2 Web3 Instance

The Web3 class is an umbrella package to house all Ethereum related modules.

```
var Web3 = require('web3');

// "Web3.providers.givenProvider" will be set if in an Ethereum supported browser.
var web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546');

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

4.3 version

Static accessible property of the Web3 class and property of the instance as well.

```
Web3.version
web3.version
```

Contains the current package version of the web3.js library.

4.3.1 Returns

`String`: The current version.

4.3.2 Example

```
web3.version;
> "1.2.3"
```

4.4 utils

Static accessible property of the Web3 class and property of the instance as well.

```
Web3.utils
web3.utils
```

Utility functions are also exposes on the `Web3` class object directly.

See *web3.utils* for more.

4.5 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
web3.bzz.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 EXCEPT `web3.bzz` which needs a separate provider at all times.

4.5.1 Parameters

1. `Object` - `myProvider`: a valid provider.

4.5.2 Returns

`Boolean`

4.5.3 Example

```
var Web3 = require('web3');
var web3 = new Web3('http://localhost:8545');
// or
var 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
var net = require('net');
var web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
```

(continues on next page)

(continued from previous page)

```
// or
var 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"
```

4.6 providers

```
web3.providers
web3.eth.providers
web3.shh.providers
web3.bzz.providers
...
```

Contains the current available providers.

4.6.1 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.

4.6.2 Example

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

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

var web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
var 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"
```

4.6.3 Configuration

```
// ====
// Http
// ====

var Web3HttpProvider = require('web3-providers-http');

var options = {
    keepAlive: true,
    withCredentials: false,
    timeout: 20000, // ms
    headers: [
        {
            name: 'Access-Control-Allow-Origin',
            value: '*'
        },
        {
            ...
        }
    ],
    agent: {
        http: http.Agent(...),
        baseUrl: ''
    }
};

var provider = new Web3HttpProvider('http://localhost:8545', options);

// ==========
// Websockets
// ==========

var Web3WsProvider = require('web3-providers-ws');

var options = {
    timeout: 30000, // ms

    // Useful for credentialed urls, e.g: ws://username:password@localhost:8546
    headers: {
      authorization: 'Basic username:password'
    },

    // Useful if requests result are large
    clientConfig: {
      maxReceivedFrameSize: 100000000,   // bytes - default: 1MiB
      maxReceivedMessageSize: 100000000, // bytes - default: 8MiB
    },

    // Enable auto reconnection
    reconnect: {
        auto: true,
        delay: 5000, // ms
        maxAttempts: 5,
        onTimeout: false
    }
};
```

(continues on next page)

(continued from previous page)

```
var ws = new Web3WsProvider('ws://localhost:8546', options);
```

More information for the Http and Websocket provider modules can be found here:

- HttpProvider
- WebsocketProvider

4.7 givenProvider

```
web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
web3.bzz.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`.

4.7.1 Returns

`Object`: The given provider set or `null`;

4.7.2 Example

4.8 currentProvider

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

Will return the current provider, otherwise `null`.

4.8.1 Returns

`Object`: The current provider set or `null`;

4.8.2 Example

4.9 BatchRequest

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

Class to create and execute batch requests.

4.9.1 Parameters

none

4.9.2 Returns

`Object`: With the following methods:

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

4.9.3 Example

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

var batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest', callback));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}, callback2));
batch.execute();
```

4.10 extend

```
web3.extend(methods)
web3.eth.extend(methods)
web3.shh.extend(methods)
web3.bzz.extend(methods)
...
```

Allows extending the web3 modules.

Note: You also have `*.extend.formatters` as additional formatter functions to be used for in and output formatting. Please see the source file for function details.

4.10.1 Parameters

1. **`methods` - `Object`: Extension object with array of methods description objects as follows:**
    - `property` - `String`: (optional) The name of the property to add to the module. If no property is set it will be added to the module directly.
    - **`methods` - `Array`: The array of method descriptions:**
        - `name` - `String`: Name of the method to add.
        - `call` - `String`: The RPC method name.
        - `params` - `Number`: (optional) The number of parameters for that function. Default 0.
        - `inputFormatter` - `Array`: (optional) Array of inputformatter functions. Each array item responds to a function parameter, so if you want some parameters not to be formatted, add a `null` instead.
        - `outputFormatter - ``Function`: (optional) Can be used to format the output of the method.

4.10.2 Returns

`Object`: The extended module.

4.10.3 Example

```
web3.extend({
    property: 'myModule',
    methods: [{
        name: 'getBalance',
        call: 'eth_getBalance',
        params: 2,
        inputFormatter: [web3.extend.formatters.inputAddressFormatter, web3.extend.
↪formatters.inputDefaultBlockNumberFormatter],
        outputFormatter: web3.utils.hexToNumberString
    },{
        name: 'getGasPriceSuperFunction',
        call: 'eth_gasPriceSuper',
        params: 2,
        inputFormatter: [null, web3.utils.numberToHex]
    }]
});

web3.extend({
    methods: [{
        name: 'directCall',
        call: 'eth_callForFun',
    }]
});

console.log(web3);
> Web3 {
    myModule: {
        getBalance: function(){},
        getGasPriceSuperFunction: function(){}
```

(continues on next page)

(continued from previous page)

```
    },
    directCall: function(){},
    eth: Eth {...},
    bzz: Bzz {...},
    ...
}
```

CHAPTER 5

web3.eth

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

```
var Eth = require('web3-eth');

// "Eth.providers.givenProvider" will be set if in an Ethereum supported browser.
var eth = new Eth(Eth.givenProvider || 'ws://some.local-or-remote.node:8546');


// or using the web3 umbrella package

var Web3 = require('web3');
var web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546');

// -> 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 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
web3.bzz.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 EXCEPT `web3.bzz` which needs a separate provider at all times.

5.10.1 Parameters

1. `Object` - `myProvider`: a valid provider.

5.10.2 Returns

`Boolean`

5.10.3 Example

```
var Web3 = require('web3');
var web3 = new Web3('http://localhost:8545');
// or
var 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
var net = require('net');
var web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
var 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"
```

5.11 providers

```
web3.providers
web3.eth.providers
web3.shh.providers
web3.bzz.providers
...
```

Contains the current available providers.

5.11.1 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.

5.11.2 Example

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

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

var web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
var 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"
```

5.11.3 Configuration

```
// ====
// Http
// ====

var Web3HttpProvider = require('web3-providers-http');

var options = {
    keepAlive: true,
    withCredentials: false,
    timeout: 20000, // ms
    headers: [
```

(continues on next page)

(continued from previous page)

```
        {
            name: 'Access-Control-Allow-Origin',
            value: '*'
        },
        {
            ...
        }
    ],
    agent: {
        http: http.Agent(...),
        baseUrl: ''
    }
};

var provider = new Web3HttpProvider('http://localhost:8545', options);

// ==========
// Websockets
// ==========

var Web3WsProvider = require('web3-providers-ws');

var options = {
    timeout: 30000, // ms

    // Useful for credentialed urls, e.g: ws://username:password@localhost:8546
    headers: {
      authorization: 'Basic username:password'
    },

    // Useful if requests result are large
    clientConfig: {
      maxReceivedFrameSize: 100000000,   // bytes - default: 1MiB
      maxReceivedMessageSize: 100000000, // bytes - default: 8MiB
    },

    // Enable auto reconnection
    reconnect: {
        auto: true,
        delay: 5000, // ms
        maxAttempts: 5,
        onTimeout: false
    }
};

var ws = new Web3WsProvider('ws://localhost:8546', options);
```

More information for the Http and Websocket provider modules can be found here:

- HttpProvider
- WebsocketProvider

5.12 givenProvider

```
web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
web3.bzz.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`.

5.12.1 Returns

`Object`: The given provider set or `null`;

5.12.2 Example

5.13 currentProvider

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

Will return the current provider, otherwise `null`.

5.13.1 Returns

`Object`: The current provider set or `null`;

5.13.2 Example

5.14 BatchRequest

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

Class to create and execute batch requests.

5.14.1 Parameters

none

5.14.2 Returns

`Object`: With the following methods:

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

5.14.3 Example

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

var batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest', callback));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}, callback2));
batch.execute();
```

5.15 extend

```
web3.extend(methods)
web3.eth.extend(methods)
web3.shh.extend(methods)
web3.bzz.extend(methods)
...
```

Allows extending the web3 modules.

Note: You also have `*.extend.formatters` as additional formatter functions to be used for in and output formatting. Please see the source file for function details.

5.15.1 Parameters

1. **`methods` - `Object`: Extension object with array of methods description objects as follows:**
    - `property` - `String`: (optional) The name of the property to add to the module. If no property is set it will be added to the module directly.
    - **`methods` - `Array`: The array of method descriptions:**
        - `name` - `String`: Name of the method to add.
        - `call` - `String`: The RPC method name.
        - `params` - `Number`: (optional) The number of parameters for that function. Default 0.
        - `inputFormatter` - `Array`: (optional) Array of inputformatter functions. Each array item responds to a function parameter, so if you want some parameters not to be formatted, add a `null` instead.
        - `outputFormatter` - ``Function`: (optional) Can be used to format the output of the method.

5.15.2 Returns

`Object`: The extended module.

5.15.3 Example

```
web3.extend({
    property: 'myModule',
    methods: [{
        name: 'getBalance',
        call: 'eth_getBalance',
        params: 2,
        inputFormatter: [web3.extend.formatters.inputAddressFormatter, web3.extend.
↪formatters.inputDefaultBlockNumberFormatter],
        outputFormatter: web3.utils.hexToNumberString
    },{
        name: 'getGasPriceSuperFunction',
        call: 'eth_gasPriceSuper',
        params: 2,
        inputFormatter: [null, web3.utils.numberToHex]
    }]
});

web3.extend({
    methods: [{
        name: 'directCall',
        call: 'eth_callForFun',
    }]
});

console.log(web3);
> Web3 {
    myModule: {
        getBalance: function(){},
        getGasPriceSuperFunction: function(){}
    },
    directCall: function(){},
    eth: Eth {...},
    bzz: Bzz {...},
    ...
}
```

5.16 defaultAccount

```
web3.eth.defaultAccount
```

This default address is used as the default `"from"` property, if no `"from"` property is specified in for the following methods:

- *web3.eth.sendTransaction()*

- web3.eth.call()
- new web3.eth.Contract() -> myContract.methods.myMethod().call()
- new web3.eth.Contract() -> myContract.methods.myMethod().send()

5.16.1 Property

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

5.16.2 Example

```
web3.eth.defaultAccount;
> undefined

// set the default account
web3.eth.defaultAccount = '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe';
```

5.17 defaultBlock

```
web3.eth.defaultBlock
```

The default block is used for certain methods. You can override it by passing in the defaultBlock as last parameter. The default value is "latest".

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

5.17.1 Property

Default block parameters can be one of the following:

- `Number|BN|BigNumber`: 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)
- `"earliest"` - `String`: The genesis block

Default is `"latest"`

5.17.2 Example

```
web3.eth.defaultBlock;
> "latest"

// set the default block
web3.eth.defaultBlock = 231;
```

5.18 defaultHardfork

```
web3.eth.defaultHardfork
```

The default hardfork property is used for signing transactions locally.

5.18.1 Property

The default hardfork property can be one of the following:

- `"chainstart"` - `String`
- `"homestead"` - `String`
- `"dao"` - `String`
- `"tangerineWhistle"` - `String`
- `"spuriousDragon"` - `String`
- `"byzantium"` - `String`
- `"constantinople"` - `String`
- `"petersburg"` - `String`
- `"istanbul"` - `String`

Default is `"petersburg"`

5.18.2 Example

```
web3.eth.defaultHardfork;
> "petersburg"

// set the default block
web3.eth.defaultHardfork = 'istanbul';
```

5.19 defaultChain

```
web3.eth.defaultChain
```

The default chain property is used for signing transactions locally.

5.19.1 Property

The default chain property can be one of the following:

- `"mainnet"` - `String`
- `"goerli"` - `String`
- `"kovan"` - `String`
- `"rinkeby"` - `String`
- `"ropsten"` - `String`

Default is `"mainnet"`

5.19.2 Example

```
web3.eth.defaultChain;
> "mainnet"

// set the default chain
web3.eth.defaultChain = 'goerli';
```

5.20 defaultCommon

```
web3.eth.defaultCommon
```

The default common property is used for signing transactions locally.

5.20.1 Property

The default common property does contain the following `Common` object:

- **`customChain` - `Object`: The custom chain properties**
  - `name` - `string`: (optional) The name of the chain
  - `networkId` - `number`: Network ID of the custom chain
  - `chainId` - `number`: Chain ID of the custom chain
- `baseChain` - `string`: (optional) `mainnet`, `goerli`, `kovan`, `rinkeby`, or `ropsten`
- `hardfork` - `string`: (optional) `chainstart`, `homestead`, `dao`, `tangerineWhistle`, `spuriousDragon`, `byzantium`, `constantinople`, `petersburg`, or `istanbul`

Default is `undefined`.

5.20.2 Example

```
web3.eth.defaultCommon;
> {customChain: {name: 'custom-network', chainId: 1, networkId: 1}, baseChain:
↪'mainnet', hardfork: 'petersburg'}
```

(continues on next page)

(continued from previous page)

```
// set the default common
web3.eth.defaultCommon = {customChain: {name: 'custom-network', chainId: 1,␣
↪networkId: 1}, baseChain: 'mainnet', hardfork: 'petersburg'};
```

5.21 transactionBlockTimeout

```
web3.eth.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.

5.21.1 Returns

`number`: The current value of transactionBlockTimeout (default: 50)

5.22 transactionConfirmationBlocks

```
web3.eth.transactionConfirmationBlocks
```

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

5.22.1 Returns

`number`: The current value of transactionConfirmationBlocks (default: 24)

5.23 transactionPollingTimeout

```
web3.eth.transactionPollingTimeout
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option defines the number of seconds Web3 will wait for a receipt which confirms that a transaction was mined by the network. NB: If this method times out, the transaction may still be pending.

5.23.1 Returns

`number`: The current value of transactionPollingTimeout (default: 750)

5.24 handleRevert

```
web3.eth.handleRevert
```

The `handleRevert` options property does default to `false` and will return the revert reason string if enabled for the following methods:

- web3.eth.call()
- *web3.eth.sendTransaction()*
- *contract.methods.myMethod(...).send(...)*
- *contract.methods.myMethod(...).call(...)*

Note: The revert reason string and the signature does exist as property on the returned error.

5.24.1 Returns

`boolean`: The current value of `handleRevert` (default: false)

5.25 getProtocolVersion

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

Returns the ethereum protocol version of the node.

5.25.1 Returns

`Promise` returns `String`: the protocol version.

5.25.2 Example

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

5.26 isSyncing

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

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

5.26.1 Returns

`Promise` returns `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

5.26.2 Example

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

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

5.27 getCoinbase

```
getCoinbase([callback])
```

Returns the coinbase address to which mining rewards will go.

5.27.1 Returns

`Promise` returns `String` - bytes 20: The coinbase address set in the node for mining rewards.

5.27.2 Example

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

5.28 isMining

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

Checks whether the node is mining or not.

5.28.1 Returns

`Promise` returns `Boolean`: `true` if the node is mining, otherwise `false`.

5.28.2 Example

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

5.29 getHashrate

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

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

5.29.1 Returns

`Promise` returns `Number`: Number of hashes per second.

5.29.2 Example

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

5.30 getGasPrice

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

Returns the current gas price oracle. The gas price is determined by the last few blocks median gas price.

5.30.1 Returns

`Promise` returns `String` - Number string of the current gas price in wei.

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

5.30.2 Example

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

5.31 getAccounts

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

Returns a list of accounts the node controls.

5.31.1 Returns

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

5.31.2 Example

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

5.32 getBlockNumber

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

Returns the current block number.

5.32.1 Returns

`Promise` returns `Number` - The number of the most recent block.

5.32.2 Example

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

5.33 getBalance

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

Get the balance of an address at a given block.

5.33.1 Parameters

1. `String` - The address to get the balance of.
2. `Number|String|BN|BigNumber` - (optional) If you pass this parameter it will not use the default block set with *web3.eth.defaultBlock*. Pre-defined block numbers as `"latest"`, `"earliest"`, `"pending"`, and `"genesis"` can also be used.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.33.2 Returns

`Promise` returns `String` - The current balance for the given address in wei.

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

5.33.3 Example

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

5.34 getStorageAt

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

Get the storage at a specific position of an address.

5.34.1 Parameters

1. `String` - The address to get the storage from.
2. `Number|String|BN|BigNumber` - The index position of the storage.
3. `Number|String|BN|BigNumber` - (optional) If you pass this parameter it will not use the default block set with *web3.eth.defaultBlock*. Pre-defined block numbers as `"latest"`, `"earliest"`, `"pending"`, and `"genesis"` can also be used.
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.34.2 Returns

`Promise` returns `String` - The value in storage at the given position.

5.34.3 Example

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

5.35 getCode

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

Get the code at a specific address.

5.35.1 Parameters

1. `String` - The address to get the code from.
2. `Number|String|BN|BigNumber` - (optional) If you pass this parameter it will not use the default block set with *web3.eth.defaultBlock*. Pre-defined block numbers as `"latest"`, `"earliest"`, `"pending"`, and `"genesis"` can also be used.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.35.2 Returns

`Promise` returns `String` - The data at given address `address`.

5.35.3 Example

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

5.36 getBlock

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

Returns a block matching the block number or block hash.

5.36.1 Parameters

1. `String|Number|BN|BigNumber` - The block number or block hash. Or the string `"genesis"`, `"latest"`, `"earliest"`, or `"pending"` as in the *default block parameter*.
2. `Boolean` - (optional, default `false`) If specified `true`, the returned block will contain all transactions as objects. By default it is `false` so, there is no need to explictly specify false. And, 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.

5.36.2 Returns

`Promise` returns `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.
- `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`: The unix timestamp for when the block was collated.
- `transactions` - `Array`: Array of transaction objects, or 32 Bytes transaction hashes depending on the `returnTransactionObjects` parameter.
- `uncles` - `Array`: Array of uncle hashes.

5.36.3 Example

```
web3.eth.getBlock(3150)
.then(console.log);

> {
    "number": 3,
    "hash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
    "parentHash": "0x2302e1c0b972d00932deb5dab9eb2982f570597d9d42504c05d9c2147eaf9c88
↪",
```

(continues on next page)

(continued from previous page)

```
    "nonce": "0xfb6e1a62d119228b",
    "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
↪",
    "logsBloom":
↪"0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "transactionsRoot":
↪"0x3a1b03875115b79539e5bd33fb00d8f7b7cd61929d5a3c574f507b8acf415bee",
    "stateRoot": "0xf1133199d44695dfa8fd1bcfe424d82854b5cebef75bddd7e40ea94cda515bcb",
    "miner": "0x8888f1f195afa192cfee860698584c030f4c9db1",
    "difficulty": '21345678965432',
    "totalDifficulty": '324567845321',
    "size": 616,
    "extraData": "0x",
    "gasLimit": 3141592,
    "gasUsed": 21662,
    "timestamp": 1429287689,
    "transactions": [
        "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b"
    ],
    "uncles": []
}
```

5.37 getBlockTransactionCount

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

Returns the number of transaction in a given block.

5.37.1 Parameters

1. `String|Number|BN|BigNumber` - The block number or hash. Or the string `"genesis"`, `"latest"`, `"earliest"`, 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.

5.37.2 Returns

`Promise` returns `Number` - The number of transactions in the given block.

5.37.3 Example

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

5.38 getBlockUncleCount

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

Returns the number of uncles in a block from a block matching the given block hash.

5.38.1 Parameters

1. `String|Number|BN|BigNumber` - The block number or hash. Or the string `"genesis"`, `"latest"`, `"earliest"`, 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.

5.38.2 Returns

`Promise` returns `Number` - The number of transactions in the given block.

5.38.3 Example

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

5.39 getUncle

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

Returns a blocks uncle by a given uncle index position.

5.39.1 Parameters

1. `String|Number|BN|BigNumber` - The block number or hash. Or the string `"genesis"`, `"latest"`, `"earliest"`, or `"pending"` as in the *default block parameter*.
2. `Number` - The index position of the uncle.
3. `Boolean` - (optional, default `false`) If specified `true`, the returned block will contain all transactions as objects. By default it is `false` so, there is no need to explictly specify false. And, if `false` it will only contains the transaction hashes.
4. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.39.2 Returns

`Promise` returns `Object` - the returned uncle. For a return value see *web3.eth.getBlock()*.

Note: An uncle doesn't contain individual transactions.

5.39.3 Example

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

5.40 getTransaction

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

Returns a transaction matching the given transaction hash.

5.40.1 Parameters

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

5.40.2 Returns

`Promise` returns `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`: Gas price provided by the sender in wei.
- `gas` - `Number`: Gas provided by the sender.
- `input` - `String`: The data sent along with the transaction.

5.40.3 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"
}
```

5.41 getPendingTransactions

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

Returns a list of pending transactions.

5.41.1 Parameters

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

5.41.2 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.

5.41.3 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',
        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'
    }
]
```

5.42 getTransactionFromBlock

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

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

5.42.1 Parameters

1. `String|Number|BN|BigNumber` - A block number or hash. Or the string `"genesis"`, `"latest"`, `"earliest"`, 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.

5.42.2 Returns

`Promise` returns `Object` - A transaction object, see *web3.eth.getTransaction*:

5.42.3 Example

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

5.43 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`.

5.43.1 Parameters

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

5.43.2 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.

5.43.3 Example

```
var receipt = web3.eth.getTransactionReceipt(
↪'0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b')
.then(console.log);

> {
  "status": true,
  "transactionHash":
↪"0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
  "transactionIndex": 0,
  "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "blockNumber": 3,
  "contractAddress": "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
  "cumulativeGasUsed": 314159,
  "gasUsed": 30234,
  "logs": [{
         // logs as returned by getPastLogs, etc.
     }, ...]
}
```

5.44 getTransactionCount

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

Get the numbers of transactions sent from this address.

5.44.1 Parameters

1. `String` - The address to get the numbers of transactions from.
2. `Number|String|BN|BigNumber` - (optional) If you pass this parameter it will not use the default block set with *web3.eth.defaultBlock*. Pre-defined block numbers as `"latest"`, `"earliest"`, `"pending"`, and `"genesis"` can also be used.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.44.2 Returns

`Promise` returns `Number` - The number of transactions sent from the given address.

5.44.3 Example

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

5.45 sendTransaction

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

Sends a transaction to the network.

5.45.1 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.
    - `chain` - `String`: (optional) Defaults to `mainnet`.
    - `hardfork` - `String`: (optional) Defaults to `petersburg`.
    - **`common` - `Object`: (optional) The common object**
        - **`customChain` - `Object`: The custom chain properties**
            - `name` - `string`: (optional) The name of the chain
            - `networkId` - `number`: Network ID of the custom chain
            - `chainId` - `number`: Chain ID of the custom chain
        - `baseChain` - `string`: (optional) `mainnet`, `goerli`, `kovan`, `rinkeby`, or `ropsten`
        - `hardfork` - `string`: (optional) `chainstart`, `homestead`, `dao`, `tangerineWhistle`, `spuriousDragon`, `byzantium`, `constantinople`, `petersburg`, or `istanbul`
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()*. If the properties `chain` and `hardfork` or `common` are not set, Web3 will try to set appropriate values by

---

querying the network for its chainId and networkId.

5.45.2 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` and `Object|undefined`: Is fired if an error occurs during sending. If the transaction was rejected by the network with a receipt, the second parameter will be the receipt.

5.45.3 Example

```
// compiled solidity source code using https://remix.ethereum.org
var code =
↪"603d80600c6000396000f3007c010000000000000000000000000000000000000000000000000000000060003504
↪";


// using the callback
web3.eth.sendTransaction({
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    data: code // deploying a contracrt
}, 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){
    ...
```

(continues on next page)

(continued from previous page)

```
})
.on('confirmation', function(confirmationNumber, receipt){ ... })
.on('error', console.error); // If a out of gas error, the second parameter is the␣
↪receipt.
```

5.46 sendSignedTransaction

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

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

5.46.1 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.

5.46.2 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.

5.46.3 Example

```
var Tx = require('ethereumjs-tx').Transaction;
var privateKey = Buffer.from(
↪'e331b6d69882b4cb4ea581d88e0b604039a3de5967688d3dcffdd2270c0fd109', 'hex');

var rawTx = {
  nonce: '0x00',
  gasPrice: '0x09184e72a000',
  gasLimit: '0x2710',
  to: '0x0000000000000000000000000000000000000000',
  value: '0x00',
  data: '0x7f7465737432000000000000000000000000000000000000000000000000000000600057'
}

var tx = new Tx(rawTx, {'chain':'ropsten'});
tx.sign(privateKey);

var serializedTx = tx.serialize();

// console.log(serializedTx.toString('hex'));
//␣
↪0xf889808609184e72a00082271094000000000000000000000000000000000000000080a47f74657374320000000000000
0

web3.eth.sendSignedTransaction('0x' + serializedTx.toString('hex'))
.on('receipt', console.log);
```

(continues on next page)

(continued from previous page)

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

Note: When use the package *ethereumjs-tx* at the version of *2.0.0*, if we don't specify the parameter *chain*, it will use *mainnet*, so if you wan to use at the other network, you should add this parameter *chain* to specify.

5.47 sign

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

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

5.47.1 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.

5.47.2 Returns

`Promise` returns `String` - The signature.

5.47.3 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
↪"
```

5.48 signTransaction

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

Signs a transaction. This account needs to be unlocked.

5.48.1 Parameters

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

5.48.2 Returns

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

5.48.3 Example

```
web3.eth.signTransaction({
    from: "0xEB014f8c8B418Db6b45774c326A0E64C78914dC0",
    gasPrice: "20000000000",
    gas: "21000",
    to: '0x3535353535353535353535353535353535353535',
    value: "1000000000000000000",
    data: ""
}).then(console.log);
> {
    raw:
↪'0xf86c808504a817c800825208943535353535353535353535353535353535353535880de0b6b3a76400008025a04f4c17
↪',
    tx: {
        nonce: '0x0',
        gasPrice: '0x4a817c800',
        gas: '0x5208',
        to: '0x3535353535353535353535353535353535353535',
        value: '0xde0b6b3a7640000',
        input: '0x',
        v: '0x25',
        r: '0x4f4c17305743700648bc4f6cd3038ec6f6af0df73e31757007b7f59df7bee88d',
        s: '0x7e1941b264348e80c78c4027afc65a87b0a5e43e86742b8ca0823584c6788fd0',
        hash: '0xda3be87732110de6c1354c83770aae630ede9ac308d9f7b399ecfba23d923384'
    }
}
```

5.49 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.

5.49.1 Parameters

1. `Object` - A transaction object, see *web3.eth.sendTransaction*. For calls the `from` property is optional however it is highly recommended to explicitly set it or it may default to *address(0)* depending on your node or provider.
2. `Number|String|BN|BigNumber` - (optional) If you pass this parameter it will not use the default block set with *web3.eth.defaultBlock*. Pre-defined block numbers as `"latest"`, `"earliest"`, `"pending"`, and `"genesis"` can also be used.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.49.2 Returns

`Promise` returns `String`: The returned data of the call, e.g. a smart contract functions return value.

5.49.3 Example

```
web3.eth.call({
    to: "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe", // contract address
    data: "0xc6888fa1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000003"
})
.then(console.log);
> "0x000000000000000000000000000000000000000000000000000000000000000a"
```

5.50 estimateGas

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

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

5.50.1 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.

5.50.2 Returns

`Promise` returns `Number` - the used gas for the simulated call/transaction.

5.50.3 Example

```
web3.eth.estimateGas({
    to: "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
    data: "0xc6888fa10000000000000000000000000000000000000000000000000000000000000003"
})
.then(console.log);
> "0x0000000000000000000000000000000000000000000000000000000000000015"
```

5.51 getPastLogs

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

Gets past logs, matching the given options.

5.51.1 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']]`

5.51.2 Returns

`Promise` returns `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.

5.51.3 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'
},{...}]
```

5.52 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").

5.52.1 Parameters

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

5.52.2 Returns

`Promise` returns `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.

5.52.3 Example

```
web3.eth.getWork()
.then(console.log);
> [
  "0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
  "0x5EED0000000000000000000000000000000000005EED0000000000000000000000000000",
  "0xd1ff1c01710000000000000000000000d1ff1c0171000000000000000000000000"
]
```

5.53 submitWork

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

Used for submitting a proof-of-work solution.

5.53.1 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.

5.53.2 Returns

`Promise` returns `Boolean` - Returns `TRUE` if the provided solution is valid, otherwise false.

5.53.3 Example

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

5.54 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

5.54.1 Parameters

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

5.54.2 Returns

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

5.54.3 Example

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

5.55 getChainId

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

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

5.55.1 Returns

`Promise<Number>` - Returns chain ID.

5.55.2 Example

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

5.56 getNodeInfo

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

5.56.1 Returns

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

5.56.2 Example

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

5.57 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.

5.57.1 Parameters

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

5.57.2 Returns

`Promise<Object>` - A account object.

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

5.57.3 Example

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

↪"0xf90211a090dcaf88c40c7bbc95a912cbdde67c175767b31173df9ee4b0d733bfdd511c43a0babe369f6b12092f49181a
↪",
```

(continues on next page)

(continued from previous page)

```
↪"0xf90211a0395d87a95873cd98c21cf1df9421af03f7247880a2554e20738eec2c7507a494a0bcf6546339a1e7e14eb8fb
↪",

↪"0xf90171a04ad705ea7bf04339fa36b124fa221379bd5a38ffe9a6112cb2d94be3a437b879a08e45b5f72e8149c01efcb
↪",

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

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.arguments`: The subscription arguments, used when re-subscribing.
- `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.
- `on("connected")` returns `String`: Fires once after the subscription successfully connected. Returns the subscription id.

6.1.3 Notification returns

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

6.1.4 Example

```
var 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`, as they use their own requestManager.

6.2.1 Parameters

1. `Boolean`: If `true` it keeps the `"syncing"` subscription.

6.2.2 Returns

`Boolean`

6.2.3 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

```
var 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.
- `"connected"` returns `Number`: Fires once after the subscription successfully connected. Returns the subscription id.

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`: The root of the receipts.
- `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.
- `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

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

        return;
    }

    console.error(error);
})
.on("connected", function(subscriptionId){
    console.log(subscriptionId);
})
.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

```
var 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. If a valid numerical `fromBlock` options property is set, Web3 will retrieve logs beginning from this point, backfilling the response as necessary.

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.
- `"connected"` returns `Number`: Fires once after the subscription successfully connected. Returns the subscription id.

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

```
var subscription = web3.eth.subscribe('logs', {
    address: '0x123456..',
    topics: ['0x12345...']
}, function(error, result){
    if (!error)
        console.log(result);
})
.on("connected", function(subscriptionId){
    console.log(subscriptionId);
})
.on("data", function(log){
    console.log(log);
})
.on("changed", function(log){
});

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

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 new 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` - `Object`: The json interface for the contract to instantiate
2. `address` - `String` (optional): The address of the smart contract to call.
3. **`options` - `Object` (optional): The options of the contract. Some are used as fallbacks for calls and transactions:**

    - `from` - `String`: The address transactions should be made from.
    - `gasPrice` - `String`: The gas price in wei to use for transactions.
    - `gas` - `Number`: The maximum gas provided for a transaction (gas limit).
    - `data` - `String`: The byte code of the contract. Used when the contract gets *deployed*.

7.1.2 Returns

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

7.1.3 Example

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

7.2 = Properties =

7.3 defaultAccount

```
web3.eth.Contract.defaultAccount
contract.defaultAccount // on contract instance
```

This default address is used as the default `"from"` property, if no `"from"` property is specified in for the following methods:

- *web3.eth.sendTransaction()*
- web3.eth.call()
- new web3.eth.Contract() -> myContract.methods.myMethod().call()
- new web3.eth.Contract() -> myContract.methods.myMethod().send()

7.3.1 Property

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

7.3.2 Example

```
web3.eth.defaultAccount;
> undefined

// set the default account
web3.eth.defaultAccount = '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe';
```

7.4 defaultBlock

```
web3.eth.Contract.defaultBlock
contract.defaultBlock // on contract instance
```

The default block is used for certain methods. You can override it by passing in the defaultBlock as last parameter. The default value of it is "latest".

7.4.1 Property

The default block parameters can be one of the following:

- `Number|BN|BigNumber`: 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)
- `"earliest"` - `String`: The genesis block

Default is `"latest"`

7.4.2 Example

```
contract.defaultBlock;
> "latest"

// set the default block
contract.defaultBlock = 231;
```

7.5 defaultHardfork

```
contract.defaultHardfork
```

The default hardfork property is used for signing transactions locally.

7.5.1 Property

The default hardfork property can be one of the following:

- `"chainstart"` - `String`
- `"homestead"` - `String`
- `"dao"` - `String`
- `"tangerineWhistle"` - `String`
- `"spuriousDragon"` - `String`
- `"byzantium"` - `String`

- `"constantinople"` - `String`
- `"petersburg"` - `String`
- `"istanbul"` - `String`

Default is `"petersburg"`

7.5.2 Example

```
contract.defaultHardfork;
> "petersburg"

// set the default block
contract.defaultHardfork = 'istanbul';
```

7.6 defaultChain

```
contract.defaultChain
```

The default chain property is used for signing transactions locally.

7.6.1 Property

The default chain property can be one of the following:

- `"mainnet"` - `String`
- `"goerli"` - `String`
- `"kovan"` - `String`
- `"rinkeby"` - `String`
- `"ropsten"` - `String`

Default is `"mainnet"`

7.6.2 Example

```
contract.defaultChain;
> "mainnet"

// set the default chain
contract.defaultChain = 'goerli';
```

7.7 defaultCommon

```
contract.defaultCommon
```

The default common property is used for signing transactions locally.

7.7.1 Property

The default common property does contain the following `Common` object:

- **`customChain` - `Object`: The custom chain properties**
  - `name` - `string`: (optional) The name of the chain
  - `networkId` - `number`: Network ID of the custom chain
  - `chainId` - `number`: Chain ID of the custom chain
- `baseChain` - `string`: (optional) `mainnet`, `goerli`, `kovan`, `rinkeby`, or `ropsten`
- `hardfork` - `string`: (optional) `chainstart`, `homestead`, `dao`, `tangerineWhistle`, `spuriousDragon`, `byzantium`, `constantinople`, `petersburg`, or `istanbul`

Default is `undefined`.

7.7.2 Example

```
contract.defaultCommon;
> {customChain: {name: 'custom-network', chainId: 1, networkId: 1}, baseChain:
↪'mainnet', hardfork: 'petersburg'}

// set the default common
contract.defaultCommon = {customChain: {name: 'custom-network', chainId: 1,␣
↪networkId: 1}, baseChain: 'mainnet', hardfork: 'petersburg'};
```

7.8 transactionBlockTimeout

```
web3.eth.Contract.transcationBlockTimeout
contract.transactionBlockTimeout // on contract instance
```

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.

7.8.1 Returns

`number`: The current value of transactionBlockTimeout (default: 50)

7.9 transactionConfirmationBlocks

```
web3.eth.Contract.transactionConfirmationBlocks
contract.transactionConfirmationBlocks // on contract instance
```

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

7.9.1 Returns

`number`: The current value of transactionConfirmationBlocks (default: 24)

7.10 transactionPollingTimeout

```
web3.eth.Contract.transactionPollingTimeout
contract.transactionPollingTimeout // on contract instance
```

The `transactionPollingTimeout` will be used over a HTTP connection. This option defines the number of seconds Web3 will wait for a receipt which confirms that a transaction was mined by the network. NB: If this method times out, the transaction may still be pending.

7.10.1 Returns

`number`: The current value of transactionPollingTimeout (default: 750)

7.11 handleRevert

```
web3.eth.Contract.handleRevert
contract.handleRevert // on contract instance
```

The `handleRevert` options property does default to `false` and will return the revert reason string if enabled on *send* or *call* of a contract method.

Note: The revert reason string and the signature does exist as property on the returned error.

7.11.1 Returns

`boolean`: The current value of `handleRevert` (default: false)

7.12 options

```
myContract.options
```

The options `object` for the contract instance. `from`, `gas` and `gasPrice` are used as fallback values when sending transactions.

7.12.1 Properties

`Object` - options:

- `address` - `String`: The address where the contract is deployed. See *options.address*.
- `jsonInterface` - `Array`: The json interface of the contract. See *options.jsonInterface*.
- `data` - `String`: The byte code of the contract. Used when the contract gets *deployed*.
- `from` - `String`: The address transactions should be made from.
- `gasPrice` - `String`: The gas price in wei to use for transactions.
- `gas` - `Number`: The maximum gas provided for a transaction (gas limit).
- `handleRevert` - `Boolean`: It will otherwise use the default value provided from the Eth module. See *handleRevert*.
- `transactionBlockTimeout` - `Number`: It will otherwise use the default value provided from the Eth module. See *transactionBlockTimeout*.
- `transactionConfirmationBlocks` - `Number`: It will otherwise use the default value provided from the Eth module. See *transactionConfirmationBlocks*.
- `transactionPollingTimeout` - `Number`: It will otherwise use the default value provided from the Eth module. See *transactionPollingTimeout*.
- `chain` - `Number`: It will otherwise use the default value provided from the Eth module. See *defaultChain*.
- `hardfork` - `Number`: It will otherwise use the default value provided from the Eth module. See *defaultHardfork*.
- `common` - `Number`: It will otherwise use the default value provided from the Eth module. See *defaultCommon*.

7.12.2 Example

```
myContract.options;
> {
    address: '0x1234567890123456789012345678901234567891',
    jsonInterface: [...],
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    gasPrice: '10000000000000',
    gas: 1000000
}

myContract.options.from = '0x1234567890123456789012345678901234567891'; // default␣
↪from address
myContract.options.gasPrice = '20000000000000'; // default gas price in wei
myContract.options.gas = 5000000; // provide as fallback always 5M gas
```

7.13 options.address

```
myContract.options.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.

7.13.1 Property

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

7.13.2 Example

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

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

7.14 options.jsonInterface

```
myContract.options.jsonInterface
```

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

7.14.1 Property

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

7.14.2 Example

```
myContract.options.jsonInterface;
> [{
    "type":"function",
    "name":"foo",
    "inputs": [{"name":"a","type":"uint256"}],
    "outputs": [{"name":"b","type":"address"}]
},{
    "type":"event",
    "name":"Event",
    "inputs": [{"name":"a","type":"uint256","indexed":true},{"name":"b","type":
↪"bytes32","indexed":false}],
}]
```

(continues on next page)

(continued from previous page)

```
// set a new interface
myContract.options.jsonInterface = [...];
```

7.15 = Methods =

7.16 clone

```
myContract.clone()
```

Clones the current contract instance.

7.16.1 Parameters

none

7.16.2 Returns

`Object`: The new contract instance.

7.16.3 Example

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

var contract2 = contract1.clone();
contract2.options.address = address2;

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

7.17 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.

7.17.1 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.

7.17.2 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.

7.17.3 Example

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

7.18 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.

7.18.1 Parameters

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

7.18.2 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.

7.18.3 Example

```
// calling a method

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

// or sending and using a promise
myContract.methods.myMethod(123).send({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
.then(function(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', function(hash){
    ...
})
.on('receipt', function(receipt){
    ...
})
.on('confirmation', function(confirmationNumber, receipt){
    ...
})
.on('error', function(error, receipt) {
    ...
});
```

7.19 methods.myMethod.call

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

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

7.19.1 Parameters

1. **`options` - `Object` (optional): The options used for calling.**
    - `from` - `String` (optional): The address the call "transaction" should be made from. For calls the `from` property is optional however it is highly recommended to explicitly set it or it may default to *address(0)* depending on your node or provider.
    - `gasPrice` - `String` (optional): The gas price in wei to use for this call "transaction".
    - `gas` - `Number` (optional): The maximum gas provided for this call "transaction" (gas limit).
2. `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.

7.19.2 Returns

`Promise` returns `Mixed`: 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:

7.19.3 Example

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

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


// MULTI-ARGUMENT RETURN:

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

// web3.js
var 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!%'
}
```

(continues on next page)

(continued from previous page)

```
// SINGLE-ARGUMENT RETURN:

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

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

7.20 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.

7.20.1 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.
    - `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.

7.20.2 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 minded.
- `"error"` returns `Error` and `Object|undefined`: Is fired if an error occurs during sending. If the transaction was rejected by the network with a receipt, the second parameter will be the receipt.

7.20.3 Example

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

// using the promise
myContract.methods.myMethod(123).send({from:
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
.then(function(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', function(hash){
    ...
})
.on('confirmation', function(confirmationNumber, receipt){
    ...
})
.on('receipt', function(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',
```

(continues on next page)

(continued from previous page)

```
                    topics: [
↪'0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385']
                },
                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', function(error, receipt) { // If the transaction was rejected by the␣
↪network with a receipt, the second parameter will be the receipt.
    ...
});
```

7.21 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. 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.

7.21.1 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.

7.21.2 Returns

`Promise` returns `Number`: The gas amount estimated.

7.21.3 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){
    ...
});
```

7.22 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.

7.22.1 Parameters

none

7.22.2 Returns

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

7.22.3 Example

```
myContract.methods.myMethod(123).encodeABI();
> '0x58cf5f10000000000000000000000000000000000000000000000000000000000000007B'
```

7.23 = Events =

7.24 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.

7.24.1 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.

7.24.2 Returns

`undefined`

7.24.3 Example

```
myContract.once('MyEvent', {
    filter: {myIndexedParam: [20,23], myOtherIndexedParam: '0x123456789...'}, //␣
↪Using an array means OR: e.g. 20 or 23
    fromBlock: 0
}, function(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,
```

(continues on next page)

(continued from previous page)

```
    transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    blockHash: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    blockNumber: 1234,
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
}
```

7.25 events

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

Subscribe to an event

7.25.1 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|String|BN|BigNumber` (optional): The block number (greater than or equal to) from which to get events on. Pre-defined block numbers as `"latest"`, `"earlist"`, `"pending"`, and `"genesis"` can also be used.
    - `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.

7.25.2 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 occours.
- `"connected"` returns `String`: Fires once after the subscription successfully connected. Returns the subscription id.

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.

7.25.3 Example

```
myContract.events.MyEvent({
    filter: {myIndexedParam: [20,23], myOtherIndexedParam: '0x123456789...'}, //␣
↪Using an array means OR: e.g. 20 or 23
    fromBlock: 0
}, function(error, event){ console.log(event); })
.on("connected", function(subscriptionId){
    console.log(subscriptionId);
})
.on('data', function(event){
    console.log(event); // same results as the optional callback above
})
.on('changed', function(event){
    // remove event from local database
})
.on('error', function(error, receipt) { // If the transaction was rejected by the␣
↪network with a receipt, the second parameter will be the receipt.
    ...
});

// 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'
}
```

7.26 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.

7.27 getPastEvents

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

Gets past events for this contract.

7.27.1 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|String|BN|BigNumber` (optional): The block number (greater than or equal to) from which to get events on. Pre-defined block numbers as `"latest"`, `"earlist"`, `"pending"`, and `"genesis"` can also be used.
    - `toBlock` - `Number|String|BN|BigNumber` (optional): The block number (less than or equal to) to get events up to (Defaults to `"latest"`). Pre-defined block numbers as `"latest"`, `"earlist"`, `"pending"`, and `"genesis"` can also be used.
    - `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.

7.27.2 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*.

7.27.3 Example

```
myContract.getPastEvents('MyEvent', {
    filter: {myIndexedParam: [20,23], myOtherIndexedParam: '0x123456789...'}, //␣
↪Using an array means OR: e.g. 20 or 23
    fromBlock: 0,
    toBlock: 'latest'
}, function(error, events){ console.log(events); })
.then(function(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',
    signature: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    logIndex: 0,
    transactionIndex: 0,
    transactionHash:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
    blockHash: '0xfd43ade1c09fade1c0d57a7af66ab4ead7c2c2eb7b11a91ffdd57a7af66ab4ead7',
    blockNumber: 1234,
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
},{
    ...
}]
```

CHAPTER 8

web3.eth.accounts

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

Note: This package has NOT been audited and might potentially be unsafe. Take precautions to clear memory properly, store the private keys safely, and test transaction receiving and sending functionality properly before using in production!

To use this package standalone use:

```
var Accounts = require('web3-eth-accounts');

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

8.1 create

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

Generates an account object with private key and public key.

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 [, ignoreLength ]);
```

Creates an account object from a private key.

For more advanced hierarchial address derivation, see [truffle-hd-awallet-provider](https://github.com/trufflesuite/truffle/tree/develop/packages/hdwallet-provider) package.

8.2.1 Parameters

1. `privateKey` - `String`: The private key to import. This is 32 bytes of random data. If you are supplying a hexadecimal number, it must have *0x* prefix in order to be in line with other Ethereum libraries. 2. `ignoreLength` - `Boolean`: If set to true does the `privateKey` length not get validated.

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 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 recevier 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.getGasPrice()*
    - `gas` - `String`: The gas provided by the transaction.
    - `chain` - `String`: (optional) Defaults to `mainnet`.
    - `hardfork` - `String`: (optional) Defaults to `petersburg`.

- **common - Object: (optional) The common object**
  - **customChain - Object: The custom chain properties**
    - name - string: (optional) The name of the chain
    - networkId - number: Network ID of the custom chain
    - chainId - number: Chain ID of the custom chain
  - baseChain - string: (optional) mainnet, goerli, kovan, rinkeby, or ropsten
  - hardfork - string: (optional) chainstart, homestead, dao, tangerineWhistle, spuriousDragon, byzantium, constantinople, petersburg, or istanbul

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: '0x31c2f03766b36f0346a850e78d4f7db2d9f4d7d54d5f272a750ba44271e370b1',
    v: '0x25',
    r: '0xc9cf86333bcb065d140032ecaab5d9281bde80f21b9687b3e94161de42d51895',
    s: '0x727a108a0b8d101465414033c3f705a9c7b826e596766046ee1183dbc8aeaa68',
    rawTransaction:
↪'0xf869808504e3b29200831e848094f0109fc8df283027b6285cc889f5aa624eac1f55843b9aca008025a0c9cf86333bcb
↪'
    transactionHash:
↪'0xde8db924885b0803d2edc335f745b2b8750c8848744905684c20b987443a9593'
}
```

(continues on next page)

(continued from previous page)

```
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'
}

// or with a common
web3.eth.accounts.signTransaction({
    to: '0xF0109fC8DF283027b6285cc889F5aA624EaC1F55',
    value: '1000000000',
    gas: 2000000
    common: {
      baseChain: 'mainnet',
      hardfork: 'petersburg',
      customChain: {
        name: 'custom-chain',
        chainId: 1,
        networkId: 1
      }
    }
}, '0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318')
.then(console.log);
```

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(
↪'0xf86180808401ef364594f0109fc8df283027b6285cc889f5aa624eac1f5580801ca031573280d608f75137e33fc1465
↪');
> "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.

8.6.1 Parameters

1. `data` - `String`: The data to sign.
2. `privateKey` - `String`: The private key to sign with.

Note: The value passed as the *data* parameter will be UTF-8 HEX decoded and wrapped as follows: `"\x19Ethereum Signed Message:\n" + message.length + message`.

8.6.2 Returns

`Object`: The signature object

- `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:
↪'0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd6007e74cd82e037b800186422fc2da1
↪'
}
```

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. `encryptedPrivateKey` - `String`: The encrypted private key to decrypt.
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(){},
```

(continues on next page)

(continued from previous page)

```
    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(){},
    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: {...},
    "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": {...}
    ...
}
```

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!

```
var Personal = require('web3-eth-personal');

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


// or using the web3 umbrella package

var Web3 = require('web3');
var web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546');

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

9.1 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
web3.bzz.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 EXCEPT `web3.bzz` which needs a separate provider at all times.

9.1.1 Parameters

1. `Object` - `myProvider`: a valid provider.

9.1.2 Returns

`Boolean`

9.1.3 Example

```
var Web3 = require('web3');
var web3 = new Web3('http://localhost:8545');
// or
var 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
var net = require('net');
var web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
var 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"
```

9.2 providers

```
web3.providers
web3.eth.providers
web3.shh.providers
web3.bzz.providers
...
```

Contains the current available providers.

9.2.1 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.

9.2.2 Example

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

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

var web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
var 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"
```

9.2.3 Configuration

```
// ====
// Http
// ====

var Web3HttpProvider = require('web3-providers-http');

var options = {
    keepAlive: true,
    withCredentials: false,
    timeout: 20000, // ms
    headers: [
        {
            name: 'Access-Control-Allow-Origin',
            value: '*'
        },
        {
            ...
        }
    ],
    agent: {
        http: http.Agent(...),
        baseUrl: ''
    }
};

var provider = new Web3HttpProvider('http://localhost:8545', options);

// ==========
// Websockets
```

(continues on next page)

(continued from previous page)

```
// ==========

var Web3WsProvider = require('web3-providers-ws');

var options = {
    timeout: 30000, // ms

    // Useful for credentialed urls, e.g: ws://username:password@localhost:8546
    headers: {
      authorization: 'Basic username:password'
    },

    // Useful if requests result are large
    clientConfig: {
      maxReceivedFrameSize: 100000000,   // bytes - default: 1MiB
      maxReceivedMessageSize: 100000000, // bytes - default: 8MiB
    },

    // Enable auto reconnection
    reconnect: {
        auto: true,
        delay: 5000, // ms
        maxAttempts: 5,
        onTimeout: false
    }
};

var ws = new Web3WsProvider('ws://localhost:8546', options);
```

More information for the Http and Websocket provider modules can be found here:

- HttpProvider
- WebsocketProvider

9.3 givenProvider

```
web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
web3.bzz.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`.

9.3.1 Returns

`Object`: The given provider set or `null`;

9.3.2 Example

9.4 currentProvider

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

Will return the current provider, otherwise `null`.

9.4.1 Returns

`Object`: The current provider set or `null`;

9.4.2 Example

9.5 BatchRequest

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

Class to create and execute batch requests.

9.5.1 Parameters

none

9.5.2 Returns

`Object`: With the following methods:

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

9.5.3 Example

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

var batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest', callback));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}, callback2));
batch.execute();
```

9.6 extend

```
web3.extend(methods)
web3.eth.extend(methods)
web3.shh.extend(methods)
web3.bzz.extend(methods)
...
```

Allows extending the web3 modules.

Note: You also have `*.extend.formatters` as additional formatter functions to be used for in and output formatting. Please see the source file for function details.

9.6.1 Parameters

1. **`methods` - `Object`: Extension object with array of methods description objects as follows:**
    - `property` - `String`: (optional) The name of the property to add to the module. If no property is set it will be added to the module directly.
    - **`methods` - `Array`: The array of method descriptions:**
        - `name` - `String`: Name of the method to add.
        - `call` - `String`: The RPC method name.
        - `params` - `Number`: (optional) The number of parameters for that function. Default 0.
        - `inputFormatter` - `Array`: (optional) Array of inputformatter functions. Each array item responds to a function parameter, so if you want some parameters not to be formatted, add a `null` instead.
        - `outputFormatter` - ``Function`: (optional) Can be used to format the output of the method.

9.6.2 Returns

`Object`: The extended module.

9.6.3 Example

```
web3.extend({
    property: 'myModule',
    methods: [{
        name: 'getBalance',
        call: 'eth_getBalance',
        params: 2,
        inputFormatter: [web3.extend.formatters.inputAddressFormatter, web3.extend.
↪formatters.inputDefaultBlockNumberFormatter],
        outputFormatter: web3.utils.hexToNumberString
    },{
        name: 'getGasPriceSuperFunction',
        call: 'eth_gasPriceSuper',
        params: 2,
        inputFormatter: [null, web3.utils.numberToHex]
    }]
});

web3.extend({
    methods: [{
        name: 'directCall',
        call: 'eth_callForFun',
    }]
});

console.log(web3);
> Web3 {
    myModule: {
        getBalance: function(){},
        getGasPriceSuperFunction: function(){}
    },
    directCall: function(){},
    eth: Eth {...},
    bzz: Bzz {...},
    ...
}
```

9.7 newAccount

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

Creates a new account.

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

9.7.1 Parameters

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

9.7.2 Returns

`Promise` returns `String`: The address of the newly created account.

9.7.3 Example

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

9.8 sign

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

The sign method calculates an Ethereum specific signature with:

```
sign(keccak256("\x19Ethereum Signed Message:\n" + dataToSign.length + dataToSign))
```

Adding a prefix to the message makes the calculated signature recognisable as an Ethereum specific signature.

If you have the original message and the signed message, you can discover the signing account address using web3.eth.personal.ecRecover (See example below)

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

9.8.1 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.

9.8.2 Returns

`Promise` returns `String` - The signature.

9.8.3 Example

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

(continues on next page)

(continued from previous page)

```
// the below is the same
web3.eth.personal.sign(web3.utils.utf8ToHex("Hello world"),
↪"0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe", "test password!")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0d
↪"

// recover the signing account address using original message and signed message
web3.eth.personal.ecRecover("Hello world", "0x30755ed65396...etc...")
.then(console.log);
> "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe"
```

9.9 ecRecover

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

Recovers the account that signed the data.

9.9.1 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.

9.9.2 Returns

`Promise` returns `String` - The account.

9.9.3 Example

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

9.10 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.

9.10.1 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.

9.10.2 Returns

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

9.10.3 Example

```
web3.eth.signTransaction({
    from: "0xEB014f8c8B418Db6b45774c326A0E64C78914dC0",
    gasPrice: "20000000000",
    gas: "21000",
    to: '0x3535353535353535353535353535353535353535',
    value: "1000000000000000000",
    data: ""
}, 'MyPassword!').then(console.log);
> {
    raw:
↪'0xf86c808504a817c800825208943535353535353535353535353535353535353535353535880de0b6b3a76400008025a04f4c1
↪',
    tx: {
        nonce: '0x0',
        gasPrice: '0x4a817c800',
        gas: '0x5208',
        to: '0x3535353535353535353535353535353535353535',
        value: '0xde0b6b3a7640000',
        input: '0x',
        v: '0x25',
        r: '0x4f4c17305743700648bc4f6cd3038ec6f6af0df73e31757007b7f59df7bee88d',
        s: '0x7e1941b264348e80c78c4027afc65a87b0a5e43e86742b8ca0823584c6788fd0',
        hash: '0xda3be87732110de6c1354c83770aae630ede9ac308d9f7b399ecfba23d923384'
    }
}
```

9.11 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.

9.11.1 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.

9.11.2 Returns

`Promise<string>` - The transaction hash.

9.11.3 Example

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

9.12 unlockAccount

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

Signs data using a specific account.

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

9.12.1 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.

9.12.2 Example

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

9.13 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.

9.13.1 Parameters

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

9.13.2 Returns

`Promise<boolean>`

9.13.3 Example

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

9.14 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`.

9.14.1 Returns

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

9.14.2 Example

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

9.15 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.

9.15.1 Parameters

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

9.15.2 Returns

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

9.15.3 Example

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

CHAPTER 10

web3.eth.ens

The `web3.eth.ens` functions let you interacting with ENS. We recommend reading the documentation ENS is providing to get deeper insights about the internals of the name service.

10.1 registryAddress

```
web3.eth.ens.registryAddress;
```

The `registryAddress` property can be used to define a custom registry address when you are connected to an unknown chain.

Note: If no address is defined will it try to detect the registry on the chain you are currently connected with and on the call of `setProvider` in the Eth module will it keep the defined address and use it for the ENS module.

10.1.1 Returns

`String` - The address of the custom registry.

10.1.2 Example

```
web3.eth.ens.registryAddress;
> "0x314159265dD8dbb310642f98f50C066173C1259b"
```

10.2 registry

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

Returns the network specific ENS registry.

10.2.1 Returns

`Registry` - The current ENS registry.

- `contract: Contract` - The `Registry` contract with the interface we know from the *Contract* object.
- `owner(name, callback): Promise` - Deprecated please use `getOwner`
- `getOwner(name, callback): Promise`
- `setOwner(name, address, txConfig, callback): PromiEvent`
- `resolver(name, callback): Promise` - Deprecated please use `getResolver`
- `getResolver(name, callback): Promise`
- `setResolver(name, address, txConfig, callback): PromiEvent`
- `getTTL(name, callback): Promise`
- `setTTL(name, ttl, txConfig, callback): PromiEvent`
- `setSubnodeOwner(name, label, address, txConfig, callback): PromiEvent`
- `setRecord(name, owner, resolver, ttl, txConfig, callback): PromiEvent`
- `setSubnodeRecord(name, label, owner, resolver, ttl, txConfig, callback): PromiEvent`
- `setApprovalForAll(operator, approved, txConfig, callback): PromiEvent`
- `isApprovedForAll(owner, operator, callback): Promise`
- `recordExists(name, callback): Promise`

10.2.2 Example

```
web3.eth.ens.registry;
> {
    contract: Contract,
    owner: Function(name, callback), // Deprecated
    getOwner: Function(name, callback),
    setOwner: Function(name, address, txConfig, callback),
    resolver: Function(name, callback), // Deprecated
    getResolver: Function(name, callback),
    setResolver: Function(name, address, txConfig, callback),
    getTTL: Function(name, callback),
    setTTL: Function(name, ttl, txConfig, callback),
    setSubnodeOwner: Function(name, label, address, txConfig, callback),
    setRecord(name, owner, resolver, ttl, txConfig, callback),
    setSubnodeRecord(name, label, owner, resolver, ttl, txConfig, callback),
    setApprovalForAll(operator, approved, txConfig, callback),
    isApprovedForAll(owner, operator, txConfig, callback),
```

(continues on next page)

(continued from previous page)

```
    recordExists(name, callback)
}
```

10.3 resolver

```
web3.eth.ens.resolver(name [, callback]);
```

Returns the resolver contract to an Ethereum address.

Note: This method is deprecated please use `getResolver`

10.3.1 Parameters

1. `name` - `String`: The ENS name.
2. `callback` - `Function`: (optional) Optional callback

10.3.2 Returns

`Promise<Resolver>` - The ENS resolver for this name.

10.3.3 Example

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

10.4 getResolver

```
web3.eth.ens.getResolver(name [, callback]);
```

Returns the resolver contract to an Ethereum address.

10.4.1 Parameters

1. `name` - `String`: The ENS name.
2. `callback` - `Function`: (optional) Optional callback

10.4.2 Returns

`Promise<Resolver>` - The ENS resolver for this name.

10.4.3 Example

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

10.5 setResolver

```
web3.eth.ens.setResolver(name, address [, txConfig ] [, callback]);
```

Does set the resolver contract address of a name.

10.5.1 Parameters

1. `name` - `String`: The ENS name.
2. `address` - `String`: The contract address of the deployed `Resolver` contract.
3. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
4. `callback` - `Function`: (optional) Optional callback

10.5.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.5.3 Example

```
web3.eth.ens.setResolver('ethereum.eth', '0x...', {...}).then(function (receipt) {
    console.log(receipt);
});
> {...}
```

10.6 getOwner

```
web3.eth.ens.getOwner(name [, callback]);
```

Returns the owner of a name.

10.6.1 Parameters

1. `name` - `String`: The ENS name.
2. `callback` - `Function`: (optional) Optional callback

10.6.2 Returns

Promise<String>‘ - The address of the registrar (EOA or CA).

10.6.3 Example

```
web3.eth.ens.getOwner('ethereum.eth').then(function (owner) {
    console.log(owner);
});
> '0x...'
```

10.7 setOwner

```
web3.eth.ens.setOwner(name [, txConfig ] [, callback]);
```

Does set the owner of the given name.

10.7.1 Parameters

1. `name` - `String`: The ENS name.
2. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
3. `callback` - `Function`: (optional) Optional callback

10.7.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.7.3 Example

```
web3.eth.ens.setOwner('ethereum.eth', {...}).then(function (receipt) {
    console.log(receipt);
});
> {...}
```

10.8 getTTL

```
web3.eth.ens.getTTL(name [, callback]);
```

Returns the caching TTL (time-to-live) of a name.

10.8.1 Parameters

1. `name` - `String`: The ENS name.
2. `callback` - `Function`: (optional) Optional callback

10.8.2 Returns

`Promise<Number>`

10.8.3 Example

```
web3.eth.ens.getTTL('ethereum.eth').then(function (ttl) {
    console.log(ttl);
});
> 100000
```

10.9 setTTL

```
web3.eth.ens.setTTL(name, ttl [, txConfig ] [, callback]);
```

Does set the caching TTL (time-to-live) of a name.

10.9.1 Parameters

1. `name` - `String`: The ENS name.
2. `ttl` - `Number`: The TTL value (uint64)
3. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here* <*eth-sendtransaction*>
4. `callback` - `Function`: (optional) Optional callback

10.9.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.9.3 Example

```
web3.eth.ens.setTTL('ethereum.eth', 10000, {...}).then(function (receipt) {
    console.log(receipt);
});
> {...}
```

10.10 setSubnodeOwner

```
web3.eth.ens.setSubnodeOwner(name, label, address [, txConfig ] [, callback]);
```

Creates a new subdomain of the given node, assigning ownership of it to the specified owner

10.10.1 Parameters

1. `name` - `String`: The ENS name.
2. `label` - `String`: The name of the sub-domain or the sha3 hash of it
3. `address` - `String`: The registrar of this sub-domain
4. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
5. `callback` - `Function`: (optional) Optional callback

10.10.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.10.3 Example

```
web3.eth.ens.setSubnodeOwner('ethereum.eth', 'web3', '0x...', {...}).then(function␣
↪(receipt) {
    console.log(receipt); // successfully defined the owner of web3.ethereum.eth
});
> {...}
```

10.11 setRecord

```
web3.eth.ens.setRecord(name, owner, resolver, ttl, [, txConfig ] [, callback]);
```

Sets the owner, resolver, and TTL for an ENS record in a single operation.

10.11.1 Parameters

1. `name` - `String`: The ENS name.
2. `owner` - `String`: The owner of the name record
3. `resolver` - `String`: The resolver address of the name record
4. `ttl` - `String | Number`: Time to live value (uint64)
5. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
6. `callback` - `Function`: (optional) Optional callback

10.11.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.11.3 Example

```
web3.eth.ens.setRecord('ethereum.eth', '0x...', '0x...', 1000000, {...}).
↪then(function (receipt) {
    console.log(receipt); // successfully registered ethereum.eth
});
> {...}
```

10.12 setSubnodeRecord

```
web3.eth.ens.setSubnodeRecord(name, label, owner, resolver, ttl, [, txConfig ] [,␣
↪callback]);
```

Sets the owner, resolver and TTL for a subdomain, creating it if necessary.

10.12.1 Parameters

1. `name` - `String`: The ENS name.
2. `label` - `String`: The name of the sub-domain or the sha3 hash of it
3. `owner` - `String`: The owner of the name record
4. `resolver` - `String`: The resolver address of the name record
5. `ttl` - `String | Number`: Time to live value (uint64)
6. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
7. `callback` - `Function`: (optional) Optional callback

10.12.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.12.3 Example

```
web3.eth.ens.setSubnodeRecord('ethereum.eth', 'web3', '0x...', '0x...', 1000000, {...}
↪).then(function (receipt) {
    console.log(receipt); // successfully registered web3.ethereum.eth
});
> {...}
```

10.13 setApprovalForAll

```
web3.eth.ens.setApprovalForAll(operator, approved, [, txConfig ] [, callback]);
```

Sets or clears an approval. Approved accounts can execute all ENS registry operations on behalf of the caller.

10.13.1 Parameters

1. `operator` - `String`: The operator address
2. `approved` - `Boolean`
3. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
4. `callback` - `Function`: (optional) Optional callback

10.13.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.13.3 Example

```
web3.eth.ens.setApprovalForAll('0x...', true, {...}).then(function (receipt) {
    console.log(receipt);
});
> {...}
```

10.14 isApprovedForAll

```
web3.eth.ens.isApprovedForAll(owner, operator [, callback]);
```

Returns `true` if the operator is approved to make ENS registry operations on behalf of the owner.

10.14.1 Parameters

1. `owner` - `String`: The owner address.
2. `operator` - `String`: The operator address.
3. `callback` - `Function`: (optional) Optional callback

10.14.2 Returns

`Promise<Boolean>`

10.14.3 Example

```
web3.eth.ens.isApprovedForAll('0x0...', '0x0...').then(function (isApproved) {
    console.log(isApproved);
})
> true
```

10.15 recordExists

```
web3.eth.ens.recordExists(name [, callback]);
```

Returns `true` if node exists in this ENS registry. This will return `false` for records that are in the legacy ENS registry but have not yet been migrated to the new one.

10.15.1 Parameters

1. `name` - `String`: The ENS name.
2. `callback` - `Function`: (optional) Optional callback

10.15.2 Returns

`Promise<Boolean>`

10.15.3 Example

```
web3.eth.ens.recordExists('0x0...', '0x0...').then(function (isExisting) {
    console.log(isExisting);
})
> true
```

10.16 getAddress

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

Resolves an ENS name to an Ethereum address.

10.16.1 Parameters

1. `ENSName` - `String`: The ENS name to resolve.
2. `callback` - `Function`: (optional) Optional callback

10.16.2 Returns

`String` - The Ethereum address of the given name.

10.16.3 Example

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

10.17 setAddress

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

Sets the address of an ENS name in his resolver.

10.17.1 Parameters

1. `ENSName` - `String`: The ENS name.
2. `address` - `String`: The address to set.
3. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
4. `callback` - `Function`: (optional) Optional callback

Emits an `AddrChanged` event.

10.17.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.17.3 Example

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

// Or using the event emitter

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

// Or listen to the AddrChanged event on the resolver

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

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

10.18 getPubkey

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

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

10.18.1 Parameters

1. `ENSName` - `String`: The ENS name.
2. `callback` - `Function`: (optional) Optional callback

10.18.2 Returns

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

10.18.3 Example

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

10.19 setPubkey

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

Sets the SECP256k1 public key associated with an ENS node

10.19.1 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. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
5. `callback` - `Function`: (optional) Optional callback

Emits an `PubkeyChanged` event.

10.19.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.19.3 Example

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

// Or using the event emitter

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

// Or listen to the PubkeyChanged event on the resolver

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

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

10.20 getContent

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

Returns the content hash associated with an ENS node.

10.20.1 Parameters

1. `ENSName` - `String`: The ENS name.
2. `callback` - `Function`: (optional) Optional callback

10.20.2 Returns

`Promise<String>` - The content hash associated with an ENS node.

10.20.3 Example

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

10.21 setContent

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

Sets the content hash associated with an ENS node.

10.21.1 Parameters

1. `ENSName` - `String`: The ENS name.
2. `hash` - `String`: The content hash to set.
3. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
4. `callback` - `Function`: (optional) Optional callback

Emits an `ContentChanged` event.

10.21.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.21.3 Example

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

// Or using the event emitter

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

// Or listen to the ContentChanged event on the resolver

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

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

10.22 getMultihash

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

Returns the multihash associated with an ENS node.

10.22.1 Parameters

1. `ENSName` - `String`: The ENS name.
2. `callback` - `Function`: (optional) Optional callback

10.22.2 Returns

`Promise<String>` - The associated multihash.

10.22.3 Example

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

10.23 supportsInterface

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

Returns `true` if the related `Resolver` does support the given signature or interfaceId.

10.23.1 Parameters

1. `name` - `String`: The ENS name.
2. `interfaceId` - `String`: The signature of the function or the interfaceId as described in the ENS documentation
3. `callback` - `Function`: (optional) Optional callback

10.23.2 Returns

`Promise<Boolean>`

10.23.3 Example

```
web3.eth.ens.supportsInterface('ethereum.eth', 'addr(bytes32').then(function (result)
↪{
    console.log(result);
});
> true
```

10.24 setMultihash

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

Sets the multihash associated with an ENS node.

10.24.1 Parameters

1. `ENSName` - `String`: The ENS name.
2. `hash` - `String`: The multihash to set.
3. `txConfig` - `Object`: (optional) The transaction options as described ::ref::*here <eth-sendtransaction>*
4. `callback` - `Function`: (optional) Optional callback

Emits an ‘‘MultihashChanged‘‘event.

10.24.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.24.3 Example

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

// Or using the event emitter

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

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

10.25 ENS events

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

10.25.1 Known resolver events

1. AddrChanged(node bytes32, a address)

2. ContentChanged(node bytes32, hash bytes32) 4. NameChanged(node bytes32, name string) 5. ABIChanged(node bytes32, contentType uint256) 6. PubkeyChanged(node bytes32, x bytes32, y bytes32)

10.25.2 Returns

`PromiEvent<TransactionReceipt | TransactionRevertInstructionError>`

10.25.3 Example

```
web3.eth.ens.resolver('ethereum.eth').then(function (resolver) {
    resolver.events.AddrChanged({fromBlock: 0}, function(error, event) {
        console.log(event);
    })
    .on('data', function(event){
        console.log(event);
    })
    .on('changed', function(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',
```

(continues on next page)

(continued from previous page)

```
    blockNumber: 1234,
    address: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'
}
```

10.25.4 Known registry events

1. Transfer(node bytes32, owner address)

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

10.25.5 Example

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

CHAPTER 11

web3.eth.Iban

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

11.1 Iban instance

This's instance of Iban

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

11.2 Iban contructor

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

Generates a iban object with conversion methods and validity checks. Also has singleton functions for conversion like Iban.toAddress(), Iban.toIban(), Iban.fromAddress(), Iban.fromBban(), Iban.createIndirect(), Iban.isValid().

11.2.1 Parameters

1. `String`: the IBAN address to instantiate an Iban instance from.

11.2.2 Returns

`Object` - The Iban instance.

11.2.3 Example

```
var iban = new web3.eth.Iban("XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS");
> Iban { _iban: 'XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS' }
```

11.3 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.

11.3.1 Parameters

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

11.3.2 Returns

`String` - The Ethereum address.

11.3.3 Example

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

11.4 toIban

static function

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

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

11.4.1 Parameters

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

11.4.2 Returns

`String` - The IBAN address.

11.4.3 Example

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

---

static function, return IBAN instance

11.5 fromAddress

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

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

11.5.1 Parameters

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

11.5.2 Returns

`Object` - The IBAN instance.

11.5.3 Example

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

---

static function, return IBAN instance

11.6 fromBban

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

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

11.6.1 Parameters

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

11.6.2 Returns

`Object` - The IBAN instance.

11.6.3 Example

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

---

static function, return IBAN instance

11.7 createIndirect

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

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

11.7.1 Parameters

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

11.7.2 Returns

`Object` - The IBAN instance.

11.7.3 Example

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

---

static function, return boolean

11.8 isValid

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

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

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

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

11.9 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.

11.9.1 Parameters

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

11.9.2 Returns

`Boolean`

11.9.3 Example

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

11.10 prototype.isDirect

method of Iban instance

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

Checks if the IBAN instance is direct.

11.10.1 Parameters

none

11.10.2 Returns

`Boolean`

11.10.3 Example

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

11.11 prototype.isIndirect

method of Iban instance

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

Checks if the IBAN instance is indirect.

11.11.1 Parameters

none

11.11.2 Returns

`Boolean`

11.11.3 Example

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

11.12 prototype.checksum

method of Iban instance

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

Returns the checksum of the IBAN instance.

11.12.1 Parameters

none

11.12.2 Returns

`String`: The checksum of the IBAN

11.12.3 Example

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

11.13 prototype.institution

method of Iban instance

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

Returns the institution of the IBAN instance.

11.13.1 Parameters

none

11.13.2 Returns

`String`: The institution of the IBAN

11.13.3 Example

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

11.14 prototype.client

method of Iban instance

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

Returns the client of the IBAN instance.

11.14.1 Parameters

none

11.14.2 Returns

`String`: The client of the IBAN

11.14.3 Example

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

11.15 prototype.toAddress

method of Iban instance

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

Returns the Ethereum address of the IBAN instance.

11.15.1 Parameters

none

11.15.2 Returns

`String`: The Ethereum address of the IBAN

11.15.3 Example

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

11.16 prototype.toString

method of Iban instance

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

Returns the IBAN address of the IBAN instance.

11.16.1 Parameters

none

11.16.2 Returns

`String`: The IBAN address.

11.16.3 Example

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

CHAPTER 12

web3.eth.abi

The `web3.eth.abi` functions let you de- and encode parameters to ABI (Application Binary Interface) for function calls to the EVM (Ethereum Virtual Machine).

12.1 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.

12.1.1 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)`

12.1.2 Returns

`String` - The ABI signature of the function.

12.1.3 Example

```
// From a JSON interface object
web3.eth.abi.encodeFunctionSignature({
    name: 'myMethod',
```

(continues on next page)

(continued from previous page)

```
    type: 'function',
    inputs: [{
        type: 'uint256',
        name: 'myNumber'
    },{
        type: 'string',
        name: 'myString'
    }]
})
> 0x24ee0097

// Or string
web3.eth.abi.encodeFunctionSignature('myMethod(uint256,string)')
> '0x24ee0097'
```

12.2 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.

12.2.1 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)`

12.2.2 Returns

`String` - The ABI signature of the event.

12.2.3 Example

```
web3.eth.abi.encodeEventSignature('myEvent(uint256,bytes32)')
> 0xf2eeb729e636a8cb783be044acf6b7b1e2c5863735b60d6daae84c366ee87d97

// or from a json interface object
web3.eth.abi.encodeEventSignature({
    name: 'myEvent',
    type: 'event',
    inputs: [{
        type: 'uint256',
        name: 'myNumber'
    },{
        type: 'bytes32',
        name: 'myBytes'
    }]
})
> 0xf2eeb729e636a8cb783be044acf6b7b1e2c5863735b60d6daae84c366ee87d97
```

12.3 encodeParameter

```
web3.eth.abi.encodeParameter(type, parameter);
```

Encodes a parameter based on its type to its ABI representation.

12.3.1 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.

12.3.2 Returns

`String` - The ABI encoded parameter.

12.3.3 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']);
>
↪"0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000
↪"

web3.eth.abi.encodeParameter(
    {
        "ParentStruct": {
            "propertyOne": 'uint256',
            "propertyTwo": 'uint256',
            "childStruct": {
                "propertyOne": 'uint256',
                "propertyTwo": 'uint256'
            }
        }
    },
    {
        "propertyOne": 42,
        "propertyTwo": 56,
        "childStruct": {
            "propertyOne": 45,
```

(continues on next page)

(continued from previous page)

```
            "propertyTwo": 78
        }
    }
);
>
↪"0x0000000000000000000000000000000000000000000000000000000000000002a0000000000000000000000000000000000
↪"
```

12.4 encodeParameters

```
web3.eth.abi.encodeParameters(typesArray, parameters);
```

Encodes a function parameters based on its *JSON interface* object.

12.4.1 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.

12.4.2 Returns

`String` - The ABI encoded parameters.

12.4.3 Example

```
web3.eth.abi.encodeParameters(['uint256','string'], ['2345675643', 'Hello!%']);
>
↪"0x0000000000000000000000000000000000000000000000000000000008bd02b7b0000000000000000000000000000000000
↪"

web3.eth.abi.encodeParameters(['uint8[]','bytes32'], [['34','434'], '0x324567fff']);
> "0x0

web3.eth.abi.encodeParameters(
    [
        'uint8[]',
        {
            "ParentStruct": {
                "propertyOne": 'uint256',
                "propertyTwo": 'uint256',
                "ChildStruct": {
                    "propertyOne": 'uint256',
                    "propertyTwo": 'uint256'
                }
            }
        }
```

(continues on next page)

(continued from previous page)

```
    ],
    [
        ['34','434'],
        {
            "propertyOne": '42',
            "propertyTwo": '56',
            "ChildStruct": {
                "propertyOne": '45',
                "propertyTwo": '78'
            }
        }
    ]
);
>
↪"0x0000000000000000000000000000000000000000000000000000000000000000000000000000000a0000000000000000000000000000000000000
↪"
```

12.5 encodeFunctionCall

```
web3.eth.abi.encodeFunctionCall(jsonInterface, parameters);
```

Encodes a function call using its *JSON interface* object and given paramaters.

12.5.1 Parameters

1. `jsonInterface` - `Object`: The *JSON interface* object of a function.
2. `parameters` - `Array`: The parameters to encode.

12.5.2 Returns

`String` - The ABI encoded function call. Means function signature + parameters.

12.5.3 Example

```
web3.eth.abi.encodeFunctionCall({
    name: 'myMethod',
    type: 'function',
    inputs: [{
        type: 'uint256',
        name: 'myNumber'
    },{
        type: 'string',
        name: 'myString'
    }]
}, ['2345675643', 'Hello!%']);
>
↪"0x24ee0097000000000000000000000000000000000000000000000000000000008bd02b7b0000000000000000000000000
↪"
```

12.6 decodeParameter

```
web3.eth.abi.decodeParameter(type, hexString);
```

Decodes an ABI encoded parameter to its JavaScript type.

12.6.1 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.

12.6.2 Returns

`Mixed` - The decoded parameter.

12.6.3 Example

```
web3.eth.abi.decodeParameter('uint256',
↪'0x0000000000000000000000000000000000000000000000000000000000000010');
> "16"

web3.eth.abi.decodeParameter('string',
↪'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪');
> "Hello!%!"

web3.eth.abi.decodeParameter('string',
↪'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪');
> "Hello!%!"

web3.eth.abi.decodeParameter(
    {
        "ParentStruct": {
          "propertyOne": 'uint256',
          "propertyTwo": 'uint256',
          "childStruct": {
            "propertyOne": 'uint256',
            "propertyTwo": 'uint256'
          }
        }
    },

,
↪'0x0000000000000000000000000000000000000000000000000000000000000002a000000000000000000000000000000000000
↪');
> {
    '0': {
```

(continues on next page)

(continued from previous page)

```
        '0': '42',
        '1': '56',
        '2': {
            '0': '45',
            '1': '78',
            'propertyOne': '45',
            'propertyTwo': '78'
        },
        'childStruct': {
            '0': '45',
            '1': '78',
            'propertyOne': '45',
            'propertyTwo': '78'
        },
        'propertyOne': '42',
        'propertyTwo': '56'
    },
    'ParentStruct': {
        '0': '42',
        '1': '56',
        '2': {
            '0': '45',
            '1': '78',
            'propertyOne': '45',
            'propertyTwo': '78'
        },
        'childStruct': {
            '0': '45',
            '1': '78',
            'propertyOne': '45',
            'propertyTwo': '78'
        },
        'propertyOne': '42',
        'propertyTwo': '56'
    }
}
```

12.7 decodeParameters

```
web3.eth.abi.decodeParameters(typesArray, hexString);
```

Decodes ABI encoded parameters to its JavaScript types.

12.7.1 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.

12.7.2 Returns

`Object` - The result object containing the decoded parameters.

12.7.3 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'
}

web3.eth.abi.decodeParameters([
  'uint8[]',
  {
    "ParentStruct": {
      "propertyOne": 'uint256',
      "propertyTwo": 'uint256',
      "childStruct": {
        "propertyOne": 'uint256',
        "propertyTwo": 'uint256'
      }
    }
  }
],
↪'0x0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000
↪');
> Result {
    '0': ['42', '24'],
    '1': {
        '0': '42',
        '1': '56',
        '2':
            {
                '0': '45',
                '1': '78',
                'propertyOne': '45',
                'propertyTwo': '78'
            },
        'childStruct':
            {
```

(continues on next page)

(continued from previous page)

```
                '0': '45',
                '1': '78',
                'propertyOne': '45',
                'propertyTwo': '78'
            },
        'propertyOne': '42',
        'propertyTwo': '56'
    }
}
```

12.8 decodeLog

```
web3.eth.abi.decodeLog(inputs, hexString, topics);
```

Decodes ABI encoded log data and indexed topic data.

12.8.1 Parameters

1. `inputs` - `Object`: 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].

12.8.2 Returns

`Object` - The result object containing the decoded parameters.

12.8.3 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 {
```

(continues on next page)

(continued from previous page)

```
    '0': 'Hello%!',
    '1': '62224',
    '2': '16',
    myString: 'Hello%!',
    myNumber: '62224',
    mySmallNumber: '16'
}
```

CHAPTER 13

web3.*.net

The `web3-net` package allows you to interact with the Ethereum nodes network properties.

```
var Net = require('web3-net');

// "Personal.providers.givenProvider" will be set if in an Ethereum supported browser.
var net = new Net(Net.givenProvider || 'ws://some.local-or-remote.node:8546');


// or using the web3 umbrella package

var Web3 = require('web3');
var web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546');

// -> web3.eth.net
// -> web3.bzz.net
// -> web3.shh.net
```

13.1 getId

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

Gets the current network ID.

13.1.1 Parameters

none

13.1.2 Returns

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

13.1.3 Example

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

13.2 isListening

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

Checks if the node is listening for peers.

13.2.1 Parameters

none

13.2.2 Returns

`Promise` returns `Boolean`

13.2.3 Example

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

13.3 getPeerCount

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

Get the number of peers connected to.

13.3.1 Parameters

none

13.3.2 Returns

`Promise` returns `Number`

13.3.3 Example

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

CHAPTER 14

web3.bzz

Note: This API might change over time.

The `web3-bzz` package allows you to interact swarm the decentralized file store. For more see the Swarm Docs.

```
var Bzz = require('web3-bzz');

// will autodetect if the "ethereum" object is present and will either connect to the␣
↪local swarm node, or the swarm-gateways.net.
// Optional you can give your own provider URL; If no provider URL is given it will␣
↪use "http://swarm-gateways.net"
var bzz = new Bzz(Bzz.givenProvider || 'http://swarm-gateways.net');


// or using the web3 umbrella package

var Web3 = require('web3');
var web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546');

// -> web3.bzz.currentProvider // if Web3.givenProvider was an ethereum provider it␣
↪will set: "http://localhost:8500" otherwise it will set: "http://swarm-gateways.net"

// set the provider manually if necessary
web3.bzz.setProvider("http://localhost:8500");
```

14.1 setProvider

```
web3.bzz.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 EXCEPT `web3.bzz` which needs a separate provider at all times.

14.1.1 Parameters

1. `Object` - `myProvider`: a valid provider.

14.1.2 Returns

`Boolean`

14.1.3 Example

```
var Bzz = require('web3-bzz');
var bzz = new Bzz('http://localhost:8500');

// change provider
bzz.setProvider('http://swarm-gateways.net');
```

14.2 givenProvider

```
web3.bzz.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`.

14.2.1 Returns

`Object`: The given provider set or `null`;

14.2.2 Example

```
bzz.givenProvider;
> {
    send: function(),
    on: function(),
    bzz: "http://localhost:8500",
    shh: true,
    ...
}

bzz.setProvider(bzz.givenProvider || "http://swarm-gateways.net");
```

14.3 currentProvider

```
bzz.currentProvider
```

Will return the current provider URL, otherwise `null`.

14.3.1 Returns

`Object`: The current provider URL or `null`;

14.3.2 Example

```
bzz.currentProvider;
> "http://localhost:8500"


if(!bzz.currentProvider) {
    bzz.setProvider("http://swarm-gateways.net");
}
```

14.4 upload

```
web3.bzz.upload(mixed)
```

Uploads files folders or raw data to swarm.

14.4.1 Parameters

1. **`mixed` - `String|Buffer|Uint8Array|Object`: The data to upload, can be a file content, file Buffer/Uint8Array, mu**

    - `String|Buffer|Uint8Array`: A file content, file Uint8Array or Buffer to upload, or:
    - **`Object`:**
        1. **Multiple key values for files and directories. The paths will be kept the same:**
            - **key must be the files path, or name, e.g. `"/foo.txt"` and its value is an object with:**

                * `type`: The mime-type of the file, e.g. `"text/html"`.
                * `data`: A file content, file Uint8Array or Buffer to upload.
        2. **Upload a file or a directory from disk in Node.js. Requires and object with the following properties:**

            - `path`: The path to the file or directory.
            - `kind`: The type of the source `"directory"`, `"file"` or `"data"`.

- `defaultFile` (optional): Path of the "defaultFile" when `"kind": "directory"`, e.g. `"/index.html"`.

3. **Upload file or folder in the browser. Requres and object with the following properties:**

- `pick`: The file picker to launch. Can be `"file"`, `"directory"` or `"data"`.

14.4.2 Returns

`Promise` returning `String`: Returns the content hash of the manifest.

14.4.3 Example

```
var bzz = web3.bzz;

// raw data
bzz.upload("test file").then(function(hash) {
    console.log("Uploaded file. Address:", hash);
})

// raw directory
var dir = {
    "/foo.txt": {type: "text/plain", data: "sample file"},
    "/bar.txt": {type: "text/plain", data: "another file"}
};
bzz.upload(dir).then(function(hash) {
    console.log("Uploaded directory. Address:", hash);
});

// upload from disk in node.js
bzz.upload({
    path: "/path/to/thing",      // path to data / file / directory
    kind: "directory",           // could also be "file" or "data"
    defaultFile: "/index.html"   // optional, and only for kind === "directory"
})
.then(console.log)
.catch(console.log);

// upload from disk in the browser
bzz.upload({pick: "file"}) // could also be "directory" or "data"
.then(console.log);
```

14.5 download

```
web3.bzz.download(bzzHash [, localpath])
```

Downloads files and folders from swarm, as buffer or to disk (only node.js).

14.5.1 Parameters

1. `bzzHash` - `String`: The file or directory to download. If the hash is a raw file it will return a Buffer, if a manifest file, it will return the directory structure. If the `localpath` is given, it will return that path where it downloaded the files to.
2. `localpath` - `String`: The local folder to download the content into. (only node.js)

14.5.2 Returns

`Promise` returning `Buffer|Object|String`: The Buffer of the file downloaded, an object with the directory structure, or the path where it was downloaded to.

14.5.3 Example

```
var bzz = web3.bzz;

// download raw file
var fileHash = "a5c10851ef054c268a2438f10a21f6efe3dc3dcdcc2ea0e6a1a7a38bf8c91e23";
bzz.download(fileHash).then(function(buffer) {
    console.log("Downloaded file:", buffer.toString());
});

// download directory, if the hash is manifest file.
var dirHash = "7e980476df218c05ecfcb0a2ca73597193a34c5a9d6da84d54e295ecd8e0c641";
bzz.download(dirHash).then(function(dir) {
    console.log("Downloaded directory:");
    > {
        'bar.txt': { type: 'text/plain', data: <Buffer 61 6e 6f 74 68 65 72 20 66 69␣
↪6c 65> },
        'foo.txt': { type: 'text/plain', data: <Buffer 73 61 6d 70 6c 65 20 66 69 6c␣
↪65> }
    }
});

// download file/directory to disk (only node.js)
var dirHash = "a5c10851ef054c268a2438f10a21f6efe3dc3dcdcc2ea0e6a1a7a38bf8c91e23";
bzz.download(dirHash, "/target/dir")
.then(path => console.log(`Downloaded directory to ${path}.`))
.catch(console.log);
```

14.6 pick

```
web3.bzz.pick.file()
web3.bzz.pick.directory()
web3.bzz.pick.data()
```

Opens a file picker in the browser to select file(s), directory or data.

14.6.1 Parameters

none

14.6.2 Returns

`Promise` returning `Object`: Returns the file or multiple files.

14.6.3 Example

```
web3.bzz.pick.file()
.then(console.log);
> {
    ...
}
```

CHAPTER 15

web3.shh

The `web3-shh` package allows you to interact with the whisper protocol for broadcasting. For more see Whisper Overview.

```
var Shh = require('web3-shh');

// "Shh.providers.givenProvider" will be set if in an Ethereum supported browser.
var shh = new Shh(Shh.givenProvider || 'ws://some.local-or-remote.node:8546');


// or using the web3 umbrella package

var Web3 = require('web3');
var web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546');

// -> web3.shh
```

15.1 setProvider

```
web3.setProvider(myProvider)
web3.eth.setProvider(myProvider)
web3.shh.setProvider(myProvider)
web3.bzz.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 EXCEPT `web3.bzz` which needs a separate provider at all times.

15.1.1 Parameters

1. `Object` - `myProvider`: a valid provider.

15.1.2 Returns

`Boolean`

15.1.3 Example

```
var Web3 = require('web3');
var web3 = new Web3('http://localhost:8545');
// or
var 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
var net = require('net');
var web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
var 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"
```

15.2 providers

```
web3.providers
web3.eth.providers
web3.shh.providers
web3.bzz.providers
...
```

Contains the current available providers.

15.2.1 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.

15.2.2 Example

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

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

var web3 = new Web3('/Users/myuser/Library/Ethereum/geth.ipc', net); // mac os path
// or
var 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"
```

15.2.3 Configuration

```
// ====
// Http
// ====

var Web3HttpProvider = require('web3-providers-http');

var options = {
    keepAlive: true,
    withCredentials: false,
    timeout: 20000, // ms
    headers: [
        {
            name: 'Access-Control-Allow-Origin',
            value: '*'
        },
        {
            ...
        }
    ],
    agent: {
        http: http.Agent(...),
        baseUrl: ''
    }
};

var provider = new Web3HttpProvider('http://localhost:8545', options);

// ==========
// Websockets
// ==========

var Web3WsProvider = require('web3-providers-ws');

var options = {
```

(continues on next page)

(continued from previous page)

```
    timeout: 30000, // ms

    // Useful for credentialed urls, e.g: ws://username:password@localhost:8546
    headers: {
      authorization: 'Basic username:password'
    },

    // Useful if requests result are large
    clientConfig: {
      maxReceivedFrameSize: 100000000,   // bytes - default: 1MiB
      maxReceivedMessageSize: 100000000, // bytes - default: 8MiB
    },

    // Enable auto reconnection
    reconnect: {
        auto: true,
        delay: 5000, // ms
        maxAttempts: 5,
        onTimeout: false
    }
};

var ws = new Web3WsProvider('ws://localhost:8546', options);
```

More information for the Http and Websocket provider modules can be found here:

- HttpProvider
- WebsocketProvider

15.3 givenProvider

```
web3.givenProvider
web3.eth.givenProvider
web3.shh.givenProvider
web3.bzz.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`.

15.3.1 Returns

`Object`: The given provider set or `null`;

15.3.2 Example

15.4 currentProvider

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

Will return the current provider, otherwise `null`.

15.4.1 Returns

`Object`: The current provider set or `null`;

15.4.2 Example

15.5 BatchRequest

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

Class to create and execute batch requests.

15.5.1 Parameters

none

15.5.2 Returns

`Object`: With the following methods:

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

15.5.3 Example

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

var batch = new web3.BatchRequest();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest', callback));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}, callback2));
batch.execute();
```

15.6 extend

```
web3.extend(methods)
web3.eth.extend(methods)
web3.shh.extend(methods)
web3.bzz.extend(methods)
...
```

Allows extending the web3 modules.

Note: You also have `*.extend.formatters` as additional formatter functions to be used for in and output formatting. Please see the source file for function details.

15.6.1 Parameters

1. **`methods` - `Object`: Extension object with array of methods description objects as follows:**
    - `property` - `String`: (optional) The name of the property to add to the module. If no property is set it will be added to the module directly.
    - **`methods` - `Array`: The array of method descriptions:**
        - `name` - `String`: Name of the method to add.
        - `call` - `String`: The RPC method name.
        - `params` - `Number`: (optional) The number of parameters for that function. Default 0.
        - `inputFormatter` - `Array`: (optional) Array of inputformatter functions. Each array item responds to a function parameter, so if you want some parameters not to be formatted, add a `null` instead.
        - `outputFormatter` - ``Function`: (optional) Can be used to format the output of the method.

15.6.2 Returns

`Object`: The extended module.

15.6.3 Example

```
web3.extend({
    property: 'myModule',
    methods: [{
        name: 'getBalance',
        call: 'eth_getBalance',
        params: 2,
        inputFormatter: [web3.extend.formatters.inputAddressFormatter, web3.extend.
↪formatters.inputDefaultBlockNumberFormatter],
        outputFormatter: web3.utils.hexToNumberString
    },{
        name: 'getGasPriceSuperFunction',
        call: 'eth_gasPriceSuper',
```

(continues on next page)

(continued from previous page)

```
        params: 2,
        inputFormatter: [null, web3.utils.numberToHex]
    }]
});

web3.extend({
    methods: [{
        name: 'directCall',
        call: 'eth_callForFun',
    }]
});

console.log(web3);
> Web3 {
    myModule: {
        getBalance: function(){},
        getGasPriceSuperFunction: function(){}
    },
    directCall: function(){},
    eth: Eth {...},
    bzz: Bzz {...},
    ...
}
```

15.7 getId

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

Gets the current network ID.

15.7.1 Parameters

none

15.7.2 Returns

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

15.7.3 Example

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

15.8 isListening

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

Checks if the node is listening for peers.

15.8.1 Parameters

none

15.8.2 Returns

`Promise` returns `Boolean`

15.8.3 Example

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

15.9 getPeerCount

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

Get the number of peers connected to.

15.9.1 Parameters

none

15.9.2 Returns

`Promise` returns `Number`

15.9.3 Example

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

15.10 getVersion

```
web3.shh.getVersion([callback])
```

Returns the version of the running whisper.

15.10.1 Parameters

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

15.10.2 Returns

`String` - The version of the current whisper running.

15.10.3 Example

```
web3.shh.getVersion()
.then(console.log);
> "5.0"
```

15.11 getInfo

```
web3.shh.getInfo([callback])
```

Gets information about the current whisper node.

15.11.1 Parameters

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

15.11.2 Returns

`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.

15.11.3 Example

```
web3.shh.getInfo()
.then(console.log);
> {
    "minPow": 0.8,
    "maxMessageSize": 12345,
    "memory": 1234335,
    "messages": 20
}
```

15.12 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).

15.12.1 Parameters

1. `Number` - Message size in bytes.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

15.12.2 Returns

`Boolean` - `true` on success, error on failure.

15.12.3 Example

```
web3.shh.setMaxMessageSize(1234565)
.then(console.log);
> true
```

15.13 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()*.

15.13.1 Parameters

1. `Number` - The new PoW requirement.
2. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

15.13.2 Returns

`Boolean` - `true` on success, error on failure.

15.13.3 Example

```
web3.shh.setMinPoW(0.9)
.then(console.log);
> true
```

15.14 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.

15.14.1 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.

15.14.2 Returns

`Boolean` - `true` on success, error on failure.

15.14.3 Example

```
web3.shh.markTrustedPeer()
.then(console.log);
> true
```

15.15 newKeyPair

```
web3.shh.newKeyPair([callback])
```

Generates a new public and private key pair for message decryption and encryption.

15.15.1 Parameters

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

15.15.2 Returns

`String` - Key ID on success and an error on failure.

15.15.3 Example

```
web3.shh.newKeyPair()
.then(console.log);
> "5e57b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f"
```

15.16 addPrivateKey

```
web3.shh.addPrivateKey(privateKey, [callback])
```

Stores a key pair derived from a private key, and returns its ID.

15.16.1 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.

15.16.2 Returns

`String` - Key ID on success and an error on failure.

15.16.3 Example

```
web3.shh.addPrivateKey(
↪'0x8bda3abeb454847b515fa9b404cede50b1cc63cfdeddd4999d074284b4c21e15')
.then(console.log);
> "3e22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f"
```

15.17 deleteKeyPair

```
web3.shh.deleteKeyPair(id, [callback])
```

Deletes the specifies key if it exists.

15.17.1 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.

15.17.2 Returns

`Boolean` - `true` on success, error on failure.

15.17.3 Example

```
web3.shh.deleteKeyPair(
↪'3e22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
> true
```

15.18 hasKeyPair

```
web3.shh.hasKeyPair(id, [callback])
```

Checks if the whisper node has a private key of a key pair matching the given ID.

15.18.1 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.

15.18.2 Returns

`Boolean` - `true` on if the key pair exist in the node, `false` if not. Error on failure.

15.18.3 Example

```
web3.shh.hasKeyPair('fe22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f
↪')
.then(console.log);
> true
```

15.19 getPublicKey

```
web3.shh.getPublicKey(id, [callback])
```

Returns the public key for a key pair ID.

15.19.1 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.

15.19.2 Returns

`String` - Public key on success and an error on failure.

15.19.3 Example

```
web3.shh.getPublicKey(
↪'3e22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
>
↪"0x04d1574d4eab8f3dde4d2dc7ed2c4d699d77cbbdd09167b8fffa099652ce4df00c4c6e0263eafe05007a46fdf0c8d32
↪"
```

15.20 getPrivateKey

```
web3.shh.getPrivateKey(id, [callback])
```

Returns the private key for a key pair ID.

15.20.1 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.

15.20.2 Returns

`String` - Private key on success and an error on failure.

15.20.3 Example

```
web3.shh.getPrivateKey(
↪'3e22b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
> "0x234234e22b9ffc2387e18636e0534534a3d0c56b0243567432453264c16e78a2adc"
```

15.21 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.

15.21.1 Parameters

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

15.21.2 Returns

`String` - Key ID on success and an error on failure.

15.21.3 Example

```
web3.shh.newSymKey()
.then(console.log);
> "cec94d139ff51d7df1d228812b90c23ec1f909afa0840ed80f1e04030bb681e4"
```

15.22 addSymKey

```
web3.shh.addSymKey(symKey, [callback])
```

Stores the key, and returns its ID.

15.22.1 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.

15.22.2 Returns

`String` - Key ID on success and an error on failure.

15.22.3 Example

```
web3.shh.addSymKey('0x5e11b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f
↪')
.then(console.log);
> "fea94d139ff51d7df1d228812b90c23ec1f909afa0840ed80f1e04030bb681e4"
```

15.23 generateSymKeyFromPassword

```
web3.shh.generateSymKeyFromPassword(password, [callback])
```

Generates the key from password, stores it, and returns its ID.

15.23.1 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.

15.23.2 Returns

`Promise<string>|undefined` - Returns the Key ID as Promise or undefined if a callback is defined.

15.23.3 Example

```
web3.shh.generateSymKeyFromPassword('Never use this password - password!')
.then(console.log);
> "2e57b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f"
```

15.24 hasSymKey

```
web3.shh.hasSymKey(id, [callback])
```

Checks if there is a symmetric key stored with the given ID.

15.24.1 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.

15.24.2 Returns

`Boolean` - `true` on if the symmetric key exist in the node, `false` if not. Error on failure.

15.24.3 Example

```
web3.shh.hasSymKey('f6dcf21ed6a17bd78d8c4c63195ab997b3b65ea683705501eae82d32667adc92')
.then(console.log);
> true
```

15.25 getSymKey

```
web3.shh.getSymKey(id, [callback])
```

Returns the symmetric key associated with the given ID.

15.25.1 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.

15.25.2 Returns

`String` - The raw symmetric key on success and an error on failure.

15.25.3 Example

```
web3.shh.getSymKey('af33b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
> "0xa82a520aff70f7a989098376e48ec128f25f767085e84d7fb995a9815eebff0a"
```

15.26 deleteSymKey

```
web3.shh.deleteSymKey(id, [callback])
```

Deletes the symmetric key associated with the given ID.

15.26.1 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.

15.26.2 Returns

`Boolean` - `true` on if the symmetric key was deleted, error on failure.

15.26.3 Example

```
web3.shh.deleteSymKey(
↪'bf31b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f')
.then(console.log);
> true
```

15.27 post

```
web3.shh.post(object [, callback])
```

This method should be called, when we want to post whisper a message to the network.

15.27.1 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.

15.27.2 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.

15.27.3 Example

```
var identities = [];
var subscription = null;

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
    subscription = shh.subscribe("messages", {
        symKeyID: identities[0],
        topics: ['0xffaadd11']
    }).on('data', console.log);

}).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(h => console.log(`Message with hash ${h} was successfuly sent`))
    .catch(err => console.log("Error: ", err));
});
```

15.28 subscribe

```
web3.shh.subscribe('messages', options [, callback])
```

Subscribe for incoming whisper messages.

15.28.1 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.

15.28.2 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.

15.28.3 Example

```
web3.shh.subscribe('messages', {
    symKeyID: 'bf31b9ffc2387e18636e0a3d0c56b023264c16e78a2adcba1303cefc685e610f',
    sig:
↪'0x04d1574d4eab8f3dde4d2dc7ed2c4d699d77cbbdd09167b8fffa099652ce4df00c4c6e0263eafe05007a46fdf0c8d32b
↪',
    ttl: 20,
    topics: ['0xffddaa11'],
    minPow: 0.8,
}, function(error, message, subscription){

    console.log(message);
    > {
        "hash": "0x4158eb81ad8e30cfcee67f20b1372983d388f1243a96e39f94fd2797b1e9c78e",
        "padding":
↪"0xc15f786f34e5cef0fef6ce7c1185d799ecdb5ebca72b3310648c5588db2e99a0d73301c7a8d90115a91213f0bc9c722
↪",
        "payload": "0xdeadbeaf",
        "pow": 0.5371803278688525,
```

(continues on next page)

(continued from previous page)

```
        "recipientPublicKey": null,
        "sig": null,
        "timestamp": 1496991876,
        "topic": "0x01020304",
        "ttl": 50
    }
})
// or
.on('data', function(message){ ... });
```

15.29 clearSubscriptions

```
web3.shh.clearSubscriptions()
```

Resets subscriptions.

Note: This will not reset subscriptions from other packages like `web3-eth`, as they use their own requestManager.

15.29.1 Parameters

1. `Boolean`: If `true` it keeps the `"syncing"` subscription.

15.29.2 Returns

`Boolean`

15.29.3 Example

```
web3.shh.subscribe('messages', {...} ,function(){ ... });

...

web3.shh.clearSubscriptions();
```

15.30 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.

15.30.1 Parameters

1. `Object`: See *web3.shh.subscribe() options* for details.

15.30.2 Returns

`String`: The filter ID.

15.30.3 Example

```
web3.shh.newMessageFilter()
.then(console.log);
> "2b47fbafb3cce24570812a82e6e93cd9e2551bbc4823f6548ff0d82d2206b326"
```

15.31 deleteMessageFilter

```
web3.shh.deleteMessageFilter(id)
```

Deletes a message filter in the node.

15.31.1 Parameters

1. `String`: The filter ID created with `shh.newMessageFilter()`.

15.31.2 Returns

`Boolean`: `true` on success, error on failure.

15.31.3 Example

```
web3.shh.deleteMessageFilter(
↪'2b47fbafb3cce24570812a82e6e93cd9e2551bbc4823f6548ff0d82d2206b326')
.then(console.log);
> true
```

15.32 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.

15.32.1 Parameters

1. `String`: The filter ID created with `shh.newMessageFilter()`.

15.32.2 Returns

`Array`: Returns an array of message objects like *web3.shh.subscribe() notification returns*

15.32.3 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
},{...}]
```

CHAPTER 16

web3.utils

This package provides utility functions for Ethereum dapps and other web3.js packages.

16.1 Bloom Filters

16.1.1 What are bloom filters?

A Bloom filter is a probabilistic, space-efficient data structure used for fast checks of set membership. That probably doesn't mean much to you yet, and so let's explore how bloom filters might be used.

Imagine that we have some large set of data, and we want to be able to quickly test if some element is currently in that set. The naive way of checking might be to query the set to see if our element is in there. That's probably fine if our data set is relatively small. Unfortunately, if our data set is really big, this search might take a while. Luckily, we have tricks to speed things up in the ethereum world!

A bloom filter is one of these tricks. The basic idea behind the Bloom filter is to hash each new element that goes into the data set, take certain bits from this hash, and then use those bits to fill in parts of a fixed-size bit array (e.g. set certain bits to 1). This bit array is called a bloom filter.

Later, when we want to check if an element is in the set, we simply hash the element and check that the right bits are in the bloom filter. If at least one of the bits is 0, then the element definitely isn't in our data set! If all of the bits are 1, then the element might be in the data set, but we need to actually query the database to be sure. So we might have false positives, but we'll never have false negatives. This can greatly reduce the number of database queries we have to make.

Real Life Example

A ethereum real life example in where this is useful is if you want to update a users balance on every new block so it stays as close to real time as possible. Without using a bloom filter on every new block you would have to force the balances even if that user may not of had any activity within that block. But if you use the logBlooms from the block you can test the bloom filter against the users ethereum address before you do any more slow operations, this will dramatically decrease the amount of calls you do as you will only be doing those extra operations if that

ethereum address is within that block (minus the false positives outcome which will be negligible). This will be highly performant for your app.

16.1.2 Functions

- web3.utils.isBloom
- web3.utils.isUserEthereumAddressInBloom
- web3.utils.isContractAddressInBloom
- web3.utils.isTopic
- web3.utils.isTopicInBloom
- web3.utils.isInBloom

Note: Please raise any issues here

16.2 randomHex

```
web3.utils.randomHex(size)
```

The randomHex library to generate cryptographically strong pseudo-random HEX strings from a given byte size.

16.2.1 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".

16.2.2 Returns

`String`: The generated random HEX string.

16.2.3 Example

```
web3.utils.randomHex(32)
> "0xa5b9d60f32436310afebcfda832817a68921beb782fabf7915cc0460b443116a"

web3.utils.randomHex(4)
> "0x6892ffc6"

web3.utils.randomHex(2)
> "0x99d6"

web3.utils.randomHex(1)
> "0x9a"
```

(continues on next page)

(continued from previous page)

```
web3.utils.randomHex(0)
> "0x"
```

16.3 _

```
web3.utils._()
```

The underscore library for many convenience JavaScript functions.

See the underscore API reference for details.

16.3.1 Example

```
var _ = web3.utils._;

_.union([1,2],[3]);
> [1,2,3]

_.each({my: 'object'}, function(value, key){ ... })

...
```

16.4 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*

16.4.1 Parameters

1. `mixed` - `String|Number`: A number, number string or HEX string to convert to a BN object.

16.4.2 Returns

`Object`: The BN.js instance.

16.4.3 Example

```
var BN = web3.utils.BN;

new BN(1234).toString();
> "1234"

new BN('1234').add(new BN('1')).toString();
> "1235"

new BN('0xea').toString();
> "234"
```

16.5 isBN

```
web3.utils.isBN(bn)
```

Checks if a given value is a BN.js instance.

16.5.1 Parameters

1. `bn` - `Object`: An BN.js instance.

16.5.2 Returns

`Boolean`

16.5.3 Example

```
var number = new BN(10);

web3.utils.isBN(number);
> true
```

16.6 isBigNumber

```
web3.utils.isBigNumber(bignumber)
```

Checks if a given value is a BigNumber.js instance.

16.6.1 Parameters

1. `bignumber` - `Object`: A BigNumber.js instance.

16.6.2 Returns

`Boolean`

16.6.3 Example

```
var number = new BigNumber(10);

web3.utils.isBigNumber(number);
> true
```

16.7 sha3

```
web3.utils.sha3(string)
web3.utils.keccak256(string) // ALIAS
```

Will calculate the sha3 of the input.

Note: To mimic the sha3 behaviour of solidity use *soliditySha3*

16.7.1 Parameters

1. `string` - `String`: A string to hash.

16.7.2 Returns

`String`: the result hash.

16.7.3 Example

```
web3.utils.sha3('234'); // taken as string
> "0xc1912fee45d61c87cc5ea59dae311904cd86b84fee17cc96966216f811ce6a79"

web3.utils.sha3(new BN('234'));
> "0xbc36789e7a1e281436464229828f817d6612f7b477d66591ff96a9e064bcc98a"

web3.utils.sha3(234);
> null // can't calculate the has of a number

web3.utils.sha3(0xea); // same as above, just the HEX representation of the number
> null

web3.utils.sha3('0xea'); // will be converted to a byte array first, and then hashed
> "0x2f20677459120677484f7104c76deb6846a2c071f9b3152c103bb12cd54d1a4a"
```

16.8 sha3Raw

```
web3.utils.sha3Raw(string)
```

Will calculate the sha3 of the input but does return the hash value instead of `null` if for example a empty string is passed.

Note: Further details about this function can be seen here *sha3*

16.9 soliditySha3

```
web3.utils.soliditySha3(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.

16.9.1 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`.

16.9.2 Returns

`String`: the result hash.

16.9.3 Example

```
web3.utils.soliditySha3('234564535', '0xfff23243', true, -10);
// auto detects:        uint256,      bytes,     bool,    int256
> "0x3e27a893dc40ef8a7f0841d96639de2f58a132be5ae466d40087a2cfa83b7179"


web3.utils.soliditySha3('Hello!%'); // auto detects: string
> "0x661136a4267dba9ccdf6bfddb7c00e714de936674c4bdb065a531cf1cb15c7fc"
```

(continues on next page)

(continued from previous page)

```
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


web3.utils.soliditySha3({t: 'string', v: 'Hello!%'}, {t: 'int8', v:-23}, {t: 'address
↪', v: '0x85F43D8a49eeB85d32Cf465507DD71d507100C1d'});
> "0xa13b31627c1ed7aaded5aecec71baf02fe123797fffd45e662eac8e06fbe4955"
```

16.10 soliditySha3Raw

```
web3.utils.soliditySha3Raw(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. The difference between this function and the `soliditySha3` function is that it will return the hash value instead of `null` if for example a empty string is given.

Note: Further details about this function can be seen here *soliditySha3*

16.11 isHex

```
web3.utils.isHex(hex)
```

Checks if a given string is a HEX string.

16.11.1 Parameters

1. `hex` - `String|HEX`: The given HEX string.

16.11.2 Returns

`Boolean`

16.11.3 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
```

16.12 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`.

16.12.1 Parameters

1. `hex` - `String|HEX`: The given HEX string.

16.12.2 Returns

`Boolean`

16.12.3 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
```

16.13 isAddress

```
web3.utils.isAddress(address)
```

Checks if a given string is a valid Ethereum address. It will also check the checksum, if the address has upper and lowercase letters.

16.13.1 Parameters

1. `address` - `String`: An address string.

16.13.2 Returns

`Boolean`

16.13.3 Example

```
web3.utils.isAddress('0xc1912fee45d61c87cc5ea59dae31190fffff232d');
> true

web3.utils.isAddress('c1912fee45d61c87cc5ea59dae31190fffff232d');
> true
```

(continues on next page)

(continued from previous page)

```
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
```

16.14 toChecksumAddress

```
web3.utils.toChecksumAddress(address)
```

Will convert an upper or lowercase Ethereum address to a checksum address.

16.14.1 Parameters

1. `address` - `String`: An address string.

16.14.2 Returns

`String`: The checksum address.

16.14.3 Example

```
web3.utils.toChecksumAddress('0xc1912fee45d61c87cc5ea59dae31190fffff232d');
> "0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d"

web3.utils.toChecksumAddress('0XC1912FEE45D61C87CC5EA59DAE31190FFFFF232D');
> "0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d" // same as above
```

16.15 checkAddressChecksum

```
web3.utils.checkAddressChecksum(address)
```

Checks the checksum of a given address. Will also return false on non-checksum addresses.

16.15.1 Parameters

1. `address` - `String`: An address string.

16.15.2 Returns

`Boolean`: `true` when the checksum of the address is valid, `false` if its not a checksum address, or the checksum is invalid.

16.15.3 Example

```
web3.utils.checkAddressChecksum('0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d');
> true
```

16.16 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.

16.16.1 Parameters

1. `mixed` - `String|Number|BN|BigNumber`: The input to convert to HEX.

16.16.2 Returns

`String`: The resulting HEX string.

16.16.3 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€');
> "0x4920686176652031303e282ac"
```

16.17 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*

16.17.1 Parameters

1. `number` - `String|Number|HEX`: Number to convert to a big number.

16.17.2 Returns

`Object`: The BN.js instance.

16.17.3 Example

```
web3.utils.toBN(1234).toString();
> "1234"

web3.utils.toBN('1234').add(web3.utils.toBN('1')).toString();
> "1235"

web3.utils.toBN('0xea').toString();
> "234"
```

16.18 hexToNumberString

```
web3.utils.hexToNumberString(hex)
```

Returns the number representation of a given HEX value as a string.

16.18.1 Parameters

1. `hexString` - `String|HEX`: A string to hash.

16.18.2 Returns

`String`: The number as a string.

16.18.3 Example

```
web3.utils.hexToNumberString('0xea');
> "234"
```

16.19 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.

16.19.1 Parameters

1. `hexString` - `String|HEX`: A string to hash.

16.19.2 Returns

`Number`

16.19.3 Example

```
web3.utils.hexToNumber('0xea');
> 234
```

16.20 numberToHex

```
web3.utils.numberToHex(number)
web3.utils.fromDecimal(number) // ALIAS, deprecated
```

Returns the HEX representation of a given number value.

16.20.1 Parameters

1. `number` - `String|Number|BN|BigNumber`: A number as string or number.

16.20.2 Returns

`String`: The HEX value of the given number.

16.20.3 Example

```
web3.utils.numberToHex('234');
> '0xea'
```

16.21 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.

16.21.1 Parameters

1. `hex` - `String`: A HEX string to convert to a UTF-8 string.

16.21.2 Returns

`String`: The UTF-8 string.

16.21.3 Example

```
web3.utils.hexToUtf8('0x4920686176652031303e282ac');
> "I have 100€"
```

16.22 hexToAscii

```
web3.utils.hexToAscii(hex)
web3.utils.toAscii(hex) // ALIAS, deprecated
```

Returns the ASCII string representation of a given HEX value.

16.22.1 Parameters

1. `hex` - `String`: A HEX string to convert to a ASCII string.

16.22.2 Returns

`String`: The ASCII string.

16.22.3 Example

```
web3.utils.hexToAscii('0x492068617665203130302021');
> "I have 100!"
```

16.23 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.

16.23.1 Parameters

1. `string` - `String`: A UTF-8 string to convert to a HEX string.

16.23.2 Returns

`String`: The HEX string.

16.23.3 Example

```
web3.utils.utf8ToHex('I have 100€');
> "0x4920686176652031303030e282ac"
```

16.24 asciiToHex

```
web3.utils.asciiToHex(string)
web3.utils.fromAscii(string) // ALIAS, deprecated
```

Returns the HEX representation of a given ASCII string.

16.24.1 Parameters

1. `string` - `String`: A ASCII string to convert to a HEX string.

16.24.2 Returns

`String`: The HEX string.

16.24.3 Example

```
web3.utils.asciiToHex('I have 100!');
> "0x4920686176652031303021"
```

16.25 hexToBytes

```
web3.utils.hexToBytes(hex)
```

Returns a byte array from the given HEX string.

16.25.1 Parameters

1. `hex` - `String|HEX`: A HEX to convert.

16.25.2 Returns

`Array`: The byte array.

16.25.3 Example

```
web3.utils.hexToBytes('0x000000ea');
> [ 0, 0, 0, 234 ]

web3.utils.hexToBytes(0x000000ea);
> [ 234 ]
```

16.26 bytesToHex

```
web3.utils.bytesToHex(byteArray)
```

Returns a HEX string from a byte array.

16.26.1 Parameters

1. `byteArray` - `Array`: A byte array to convert.

16.26.2 Returns

`String`: The HEX string.

16.26.3 Example

```
web3.utils.bytesToHex([ 72, 101, 108, 108, 111, 33, 36 ]);
> "0x48656c6c6f2125"
```

16.27 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.

16.27.1 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'

16.27.2 Returns

`String|BN`: If a string is given it returns a number string, otherwise a BN.js instance.

16.27.3 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"
```

16.28 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.

16.28.1 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'

16.28.2 Returns

`String`: It always returns a string number.

16.28.3 Example

```
web3.utils.fromWei('1', 'ether');
> "0.000000000000000001"

web3.utils.fromWei('1', 'finney');
> "0.000000000000001"

web3.utils.fromWei('1', 'szabo');
> "0.000000000001"
```

(continues on next page)

(continued from previous page)

```
web3.utils.fromWei('1', 'shannon');
> "0.000000001"
```

16.29 unitMap

```
web3.utils.unitMap
```

Shows all possible ether value and their amount in wei.

16.29.1 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`: '10000000000000000000'
- `mether`: '10000000000000000000000000'
- `gether`: '10000000000000000000000000000'
- `tether`: '10000000000000000000000000000000'

16.29.2 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:       '10000000000000000000',
    kether:      '10000000000000000000000',
    grand:       '10000000000000000000000',
    mether:      '10000000000000000000000000',
    gether:      '10000000000000000000000000000',
    tether:      '10000000000000000000000000000000'
}
```

16.30 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.

16.30.1 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"`.

16.30.2 Returns

`String`: The padded string.

16.30.3 Example

```
web3.utils.padLeft('0x3456ff', 20);
> "0x0000000000000003456ff"

web3.utils.padLeft(0x3456ff, 20);
> "0x0000000000000003456ff"

web3.utils.padLeft('Hello', 20, 'x');
> "xxxxxxxxxxxxxxxHello"
```

16.31 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.

16.31.1 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"`.

16.31.2 Returns

`String`: The padded string.

16.31.3 Example

```
web3.utils.padRight('0x3456ff', 20);
> "0x3456ff00000000000000"

web3.utils.padRight(0x3456ff, 20);
> "0x3456ff00000000000000"

web3.utils.padRight('Hello', 20, 'x');
> "Helloxxxxxxxxxxxxxxx"
```

16.32 toTwosComplement

```
web3.utils.toTwosComplement(number)
```

Converts a negative numer into a two's complement.

16.32.1 Parameters

1. `number` - `Number|String|BigNumber`: The number to convert.

16.32.2 Returns

`String`: The converted hex string.

16.32.3 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"
```

Index