app3.js Documentation

Release 0.0.1

APIS

Jul 04, 2019

User Documentation

1 Getting Started **3**
- 1.1 Adding app3js . . . 3

2 Callbacks Promises Events **5**

3 Glossary **7**
- 3.1 json interface . . . 7

4 App3 **9**
- 4.1 App3.modules . . . 9
- 4.2 app3 object . . . 10
- 4.3 version . . . 10
- 4.4 utils . . . 11
- 4.5 setProvider . . . 11
- 4.6 providers . . . 11
- 4.7 givenProvider . . . 12
- 4.8 currentProvider . . . 12
- 4.9 BatchRequest . . . 13
- 4.10 extend . . . 13

5 app3.apis **17**
- 5.1 Note on checksum addresses . . . 17
- 5.2 subscribe . . . 18
- 5.3 Contract . . . 18
- 5.4 personal . . . 18
- 5.5 accounts . . . 18
- 5.6 abi . . . 18
- 5.7 net . . . 18
- 5.8 defaultAccount . . . 18
- 5.9 defaultBlock . . . 19
- 5.10 getProtocolVersion . . . 20
- 5.11 isSyncing . . . 20
- 5.12 getCoinbase . . . 21
- 5.13 isMining . . . 21
- 5.14 getGasPrice . . . 21
- 5.15 getAccounts . . . 22
- 5.16 getWalletInfo . . . 23

5.17 getBlockNumber . . . 24
5.18 getBalance . . . 24
5.19 getCode . . . 25
5.20 getBlock . . . 25
5.21 getBlockTransactionCount . . . 27
5.22 getTransaction . . . 28
5.23 getTransactionsByKeyword . . . 29
5.24 getRecentTransactions . . . 31
5.25 getRecentBlocks . . . 32
5.26 getTransactionFromBlock . . . 34
5.27 getTransactionReceipt . . . 35
5.28 getTransactionCount . . . 37
5.29 getMasternodeCount . . . 37
5.30 sendTransaction . . . 38
5.31 sendSignedTransaction . . . 40
5.32 sign . . . 41
5.33 call . . . 42
5.34 estimateGas . . . 42
5.35 getPastLogs . . . 43

6 app3.apis.subscribe **45**
6.1 subscribe . . . 45
6.2 clearSubscriptions . . . 46
6.3 subscribe(“pendingTransactions”) . . . 47
6.4 subscribe(“newBlockHeaders”) . . . 47
6.5 subscribe(“logs”) . . . 49

7 app3.apis.Contract **51**
7.1 new contract . . . 51
7.2 = Properties = . . . 52
7.3 options . . . 52
7.4 options.address . . . 53
7.5 options.jsonInterface . . . 53
7.6 = Methods = . . . 54
7.7 clone . . . 54
7.8 deploy . . . 55
7.9 methods . . . 56
7.10 methods.myMethod.call . . . 57
7.11 methods.myMethod.send . . . 59
7.12 methods.myMethod.estimateGas . . . 61
7.13 methods.myMethod.encodeABI . . . 62
7.14 = Events = . . . 63
7.15 once . . . 63
7.16 events . . . 64
7.17 events.allEvents . . . 65
7.18 getPastEvents . . . 66

8 app3.apis.accounts **69**
8.1 create . . . 69
8.2 privateKeyToAccount . . . 70
8.3 signTransaction . . . 71
8.4 authTransaction . . . 73
8.5 recoverTransaction . . . 74
8.6 hashMessage . . . 74

8.7 sign . . . 75
8.8 recover . . . 76
8.9 encrypt . . . 77
8.10 decrypt . . . 78
8.11 wallet . . . 79
8.12 wallet.create . . . 79
8.13 wallet.add . . . 80
8.14 wallet.remove . . . 81
8.15 wallet.clear . . . 82
8.16 wallet.encrypt . . . 82
8.17 wallet.decrypt . . . 83
8.18 wallet.save . . . 84
8.19 wallet.load . . . 85

9 app3.apis.personal **87**
9.1 newAccount . . . 87
9.2 sign . . . 88
9.3 ecRecover . . . 89
9.4 signTransaction . . . 89
9.5 sendTransaction . . . 90
9.6 unlockAccount . . . 91
9.7 lockAccount . . . 92
9.8 getAccounts . . . 92
9.9 importRawKey . . . 93

10 app3.apis.abi **95**
10.1 encodeFunctionSignature . . . 95
10.2 encodeEventSignature . . . 96
10.3 encodeParameter . . . 97
10.4 encodeParameters . . . 98
10.5 encodeFunctionCall . . . 99
10.6 decodeParameter . . . 100
10.7 decodeParameters . . . 101
10.8 decodeLog . . . 103

11 API **105**
11.1 Setting up APIS Core HTTP RPC . . . 105
11.2 Securing Your Credentials . . . 107
11.3 Make Requests . . . 107
11.4 apis_protocolVersion . . . 107
11.5 apis_syncing . . . 108
11.6 apis_coinbase . . . 108
11.7 apis_mining . . . 109
11.8 apis_gasPrice . . . 109
11.9 apis_accounts . . . 110
11.10 apis_getWalletInfo . . . 111
11.11 apis_blockNumber . . . 112
11.12 apis_getBalance . . . 112
11.13 apis_getCode . . . 113
11.14 apis_getBlockByNumber . . . 114
11.15 apis_getBlockTransactionCountByNumber . . . 116
11.16 apis_getTransactionByHash . . . 116
11.17 apis_getTransactionsByKeyword . . . 117
11.18 apis_getRecentTransactions . . . 118

11.19 apis_getRecentBlocks . . . 119
11.20 apis_getTransactionByBlockNumberAndIndex . . . 121
11.21 apis_getTransactionByBlockHashAndIndex . . . 122
11.22 apis_getTransactionReceipt . . . 123
11.23 apis_getTransactionCount . . . 125
11.24 apis_getMasternodeCount . . . 126
11.25 apis_sendRawTransaction . . . 127
11.26 apis_sendTransaction . . . 127
11.27 apis_sign . . . 128
11.28 apis_call . . . 129
11.29 apis_estimateGas . . . 130
11.30 apis_getLogs . . . 130
11.31 apis_call . . . 132
11.32 personal_sign . . . 133
11.33 personal_ecRecover . . . 134
11.34 personal_signTransaction . . . 134
11.35 personal_unlockAccount . . . 135
11.36 personal_lockAccount . . . 136
11.37 personal_listAccounts . . . 137
11.38 personal_importRawKey . . . 137

Index **139**

app3js is a collection of libraries which allow you to interact with a local or remote APIS node, using a WebSocket connection.

The following documentation will guide you through *installing and running app3js*, as well as providing a API reference documentation with examples.

CHAPTER 1

Getting Started

The app3js library is a collection of modules which contain specific functionality for the APIS ecosystem.

- The `app3-apis` is for the APIS blockchain and smart contracts
- The `app3-utils` contains useful helper functions for Dapp developers.

1.1 Adding app3js

First you need to get app3js into your project. This can be done using the following methods:

- npm: `npm install app3js`

After that you need to create a app3js instance and set a provider. You should connect to a remote/local node.

```
// in node.js use: var App3 = require('app3js');

const wsProvider = new App3.providers.WebsocketProvider('ws://
↪153a17085797541e1b821aa7f0:111d8060fd25b75dfadbe0379f@127.0.0.1:40405');

// Disable AES encryption and improve communication speed with RPC.
wsProvider.enableEncryption(false);

var app3 = new App3(App3.givenProvider || wsProvider);
```

```
// in node.js use: var App3 = require('app3js');

const httpProvider = new new App3.providers.HttpProvider('http://
↪a6180fb19186e1e14a83cadad92d13c1:4bc974a374d9552a30faa2269e5dd3a6@127.0.0.1:47194');

const app3 = new App3();

app3.setProvider(httpProvider);
```

That's it! now you can use the `app3` object.

CHAPTER 2

Callbacks Promises Events

To help app3 integrate into all kind of projects with different standards we provide multiple ways to act on asynchronous functions.

Most app3js objects allow a callback as the last parameter, as well as returning promises to chain functions.

APIS 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 `app3.apis.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".

```
app3.apis.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 APIS smart contract.

Using this json interface app3js is able to create JavaScript object representing the smart contract and its methods and events using the *app3.apis.Contract object*.

3.1.1 Specification

Functions:

- `type`: `"function"`, `"constructor"` (can be omitted, defaulting to `"function"`; `"fallback"` also possible but not relevant in app3js);
- `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

App3

Class

This's main class of anything related APIS.

```
var App3 = require('app3js');

> App3.utils
> App3.version
> App3.providers
> App3.modules
```

4.1 App3.modules

Property of App3 class

```
App3.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 modules:

- `Apis` - `Function`: the Apis module for interacting with the APIS network see *app3.apis* for more.
- `Net` - `Function`: the Net module for interacting with network properties see app3.apis.net for more.
- `Personal` - `Function`: the Personal module for interacting with the APIS accounts see app3.apis.personal for more.

4.1.2 Example

```
App3.modules
> {
    Apis: Apis function(provider),
    Net: Net function(provider),
    Personal: Personal function(provider)
}
```

4.2 app3 object

The instance of App3

The app3js object is an umbrella package to house all APIS related modules.

```
var App3 = require('app3js');

var app3 = new App3('ws://some.local-or-remote.node:8546');

> app3.apis
> app3.utils
> app3.version
```

4.3 version

Property of App3 class and instance of App3

```
App3.version
app3.version
```

Contains the version of the `app3` container object.

4.3.1 Returns

`String`: The current version.

4.3.2 Example

```
app3.version;
> "0.9.3-6"
```

4.4 utils

Property of App3 class and instance of App3

```
App3.utils
app3.utils
```

Utility functions are also exposes on the `App3` class object directly.

See app3.utils for more.

4.5 setProvider

```
app3.setProvider(myProvider)
app3.apis.setProvider(myProvider)
...
```

Will change the provider for its module.

Note: When called on the umbrella package `app3` it will also set the provider for all sub modules `web3.apis`, `web3.shh`, etc 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 App3 = require('app3js');

app3.setProvider('ws://localhost:8546');
// or
app3.setProvider(new App3.providers.WebsocketProvider('ws://localhost:8546'));
```

4.6 providers

```
app3.providers
app3.apis.providers
...
```

Contains the current available providers.

4.6.1 Value

`Object` with the following providers:

- `Object` - `WebsocketProvider`: The Websocket provider is the standard for usage in legacy browsers.

4.6.2 Example

```
var App3 = require('app3');
var app3 = new App3('ws://remotenode.com:8546');
// or
var app3 = new App3(new App3.providers.WebsocketProvider('ws://remotenode.com:8546'));
```

4.7 givenProvider

```
app3.givenProvider
app3.apis.givenProvider
...
```

When using app3js in an APIS 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

```
app3.currentProvider
app3.apis.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 app3.BatchRequest()
new app3.apis.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 app3.apis.Contract(abi, address);

var batch = new app3.BatchRequest();
batch.add(app3.apis.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest', callback));
batch.add(contract.methods.balance(address).call.request({from:
↪'0x0000000000000000000000000000000000000000'}, callback2));
batch.execute();
```

4.10 extend

```
app3.extend(methods)
app3.apis.extend(methods)
...
```

Allows extending the app3 modules.

Note: You also have `*.extend.formatters` as additional formatter functions to be used for in and output formatting.

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

```
app3.extend({
    property: 'myModule',
    methods: [{
        name: 'getBalance',
        call: 'apis_getBalance',
        params: 2,
        inputFormatter: [app3.extend.formatters.inputAddressFormatter, app3.extend.
↪formatters.inputDefaultBlockNumberFormatter],
        outputFormatter: app3.utils.hexToNumberString
    },{
        name: 'getGasPriceSuperFunction',
        call: 'apis_gasPriceSuper',
        params: 2,
        inputFormatter: [null, app3.utils.numberToHex]
    }]
});

app3.extend({
    methods: [{
        name: 'directCall',
        call: 'apis_callForFun',
    }]
});

console.log(app3);
> App3 {
    myModule: {
        getBalance: function(){},
        getGasPriceSuperFunction: function(){}
```

(continues on next page)

(continued from previous page)

```
    },
    directCall: function(){},
    ...
}
```

CHAPTER 5

app3.apis

The `app3-apis` package allows you to interact with an APIS blockchain and APIS smart contracts.

```
var Apis = require('app3-apis');
var apis = new Apis('ws://some.local-or-remote.node:8546');


// or using the app3 umbrella package

var App3 = require('app3');
var app3 = new App3('ws://some.local-or-remote.node:8546');

// -> app3.apis
```

5.1 Note on checksum addresses

All APIS 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

```
app3.apis.getAccounts(console.log);
> ["0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe" ,
↪"0x85F43D8a49eeB85d32Cf465507DD71d507100C1d"]
```

5.2 subscribe

For `app3.apis.subscribe` see the *Subscribe reference documentation*

5.3 Contract

For `app3.apis.Contract` see the *Contract reference documentation*

5.4 personal

For `app3.apis.personal` see the personal reference documentation

5.5 accounts

For `app3.apis.accounts` see the *accounts reference documentation*

5.6 abi

For `app3.apis.abi` see the *ABI reference documentation*

5.7 net

For `app3.apis.net` see the net reference documentation

5.8 defaultAccount

```
app3.apis.defaultAccount
```

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

- *app3.apis.sendTransaction()*
- app3.apis.call()
- new app3.apis.Contract() -> myContract.methods.myMethod().call()
- new app3.apis.Contract() -> myContract.methods.myMethod().send()

5.8.1 Property

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

5.8.2 Example

```
app3.apis.defaultAccount;
> undefined

// set the default account
app3.apis.defaultAccount = '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe';
```

5.9 defaultBlock

```
app3.apis.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".

- app3.apis.getBalance()
- app3.apis.getCode()
- *app3.apis.getTransactionCount()*
- app3.apis.getStorageAt()
- app3.apis.call()
- new app3.apis.Contract() -> myContract.methods.myMethod().call()

5.9.1 Property

Default block parameters can be one of the following:

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

Default is `"latest"`

5.9.2 Example

```
app3.apis.defaultBlock;
> "latest"

// set the default block
app3.apis.defaultBlock = 231;
```

5.10 getProtocolVersion

```
app3.apis.getProtocolVersion([callback])
```

Returns the APIS protocol version of the node.

5.10.1 Returns

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

5.10.2 Example

```
app3.apis.getProtocolVersion()
.then(console.log);
> "63"
```

5.11 isSyncing

```
app3.apis.isSyncing([callback])
```

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

5.11.1 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.

5.11.2 Example

```
app3.apis.isSyncing()
.then(console.log);

> {
    startingBlock: 40606,
    currentBlock: 40612,
    highestBlock: 40612,
}
```

5.12 getCoinbase

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

Returns the coinbase address to which mining rewards will go.

5.12.1 Returns

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

5.12.2 Example

```
app3.apis.getCoinbase()
.then(console.log);
> "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe"
```

5.13 isMining

```
app3.apis.isMining([callback])
```

Checks whapiser the node is mining or not.

5.13.1 Returns

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

5.13.2 Example

```
app3.apis.isMining()
.then(console.log);
> true
```

5.14 getGasPrice

```
app3.apis.getGasPrice([callback])
```

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

5.14.1 Returns

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

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

5.14.2 Example

```
app3.apis.getGasPrice()
.then(console.log);
> "50000000000"
```

5.15 getAccounts

```
app3.apis.getAccounts([callback])
```

Returns a list of accounts the node controls.

5.15.1 Returns

`Promise` returns `Array` - Array of accounts controlled by node objects.

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

- `address` 32 Bytes - `String`: address of account
- `index` - `Number`: The index position of accounts controlled by node.
- `aAPIS` - `Number`: The current APIS balance for the given account in atto.
- `aMNR` - `Number`: The current MNR balance for the given account in atto.
- `nonce` - `Number`: The number of transactions
- `APIS` 32 Bytes - `String`: The current APIS balance for the given account.
- `MNR` 32 Bytes - `String`: The current MNR balance for the given account.
- `proofKey` 32 Bytes - `String`: 2-Step Verification Key. `null` if not registered.
- `isMasternode` = `Boolean`: True if given address is a masternode.

5.15.2 Example

```
app3.apis.getAccounts()
.then(console.log);
> [ { address: '0x2947e8f4822fef47241d619910050a5c3660c0b9',
      index: 0,
      aAPIS: '1551344056726774550000000',
      aMNR: '20000000000000000',
      nonce: '0',
      APIS: '1_551_344.05672677455',
      MNR: '0.02' },
    { address: '0x036684e72c49c0121823c647587bbd7676d0b998',
      index: 1,
      aAPIS: '0',
      aMNR: '0',
      nonce: '1',
      APIS: '0',
      MNR: '0',
      proofKey: '0x700dc8874a7e187a0e259e6293d839b98ea5c6a9'} ]
```

5.16 getWalletInfo

```
app3.apis.getWalletInfo(addressOrMask [, callback])
```

Returns a information of given address.

5.16.1 Parameters

1. `String` - The address to get the information. Or the string mask like "`some_name@some_domain`"

5.16.2 Returns

`Promise` returns `Object` - Information of given address.

- `address` 32 Bytes - `String`: address of account.
- `mask` - `String`: Mask of given address. `null` if not registered.
- `aAPIS` - `Number`: The current APIS balance for the given account in atto.
- `aMNR` - `Number`: The current MNR balance for the given account in atto.
- `nonce` - `Number`: The number of transactions
- `APIS` 32 Bytes - `String`: The current APIS balance for the given account.
- `MNR` 32 Bytes - `String`: The current MNR balance for the given account.
- `proofKey` 32 Bytes - `String`: 2-Step Verification Key. `null` if not registered.
- `isContract` - `String`: True if given address has contract code. `null` if hasn't.
- `isMasternode` = `Boolean`: True if given address is a masternode.

5.16.3 Example

```
app3.apis.getWalletInfo('0xea31b942f886fcbbcfedd5580f992afe464a38b8');
// Or app3.apis.getWalletInfo('Daryl@me')
.then(console.log);
> { address: '0xea31b942f886fcbbcfedd5580f992afe464a38b8',
    mask: 'Daryl@me',
    aAPIS: '453684918145940000000000',
    aMNR: '627846000000000',
    nonce: '5',
    APIS: '45_368.491814594',
    MNR: '0.000627846',
    isMasternode: false }
```

5.17 getBlockNumber

```
app3.apis.getBlockNumber([callback])
```

Returns the current block number.

5.17.1 Returns

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

5.17.2 Example

```
app3.apis.getBlockNumber()
.then(console.log);
> 2744
```

5.18 getBalance

```
app3.apis.getBalance(address [, defaultBlock] [, callback])
```

Get the balance of an address at a given block.

5.18.1 Parameters

1. `String` - The address to get the balance of.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *app3.apis.defaultBlock*.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.18.2 Returns

`Promise` returns `Object` - The current balance for the given address in atto.

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

- `aAPIS` - `Number`: The current APIS balance for the given account in atto.
- `aMNR` - `Number`: The current MNR balance for the given account in atto.
- `APIS` - `Number`: The current readable APIS balance for the given account.
- `MNR` - `Number`: The current readable MNR balance for the given account.

5.18.3 Example

```
app3.apis.getBalance("0x407d73d8a49eeb85d32cf465507dd71d507100c1")
.then(console.log);
> { attoAPIS: '1553284456726774550000000',
    attoMNR: '20000000000000000',
    APIS: '1_553_284.45672677455',
    MNR: '0.02' }
```

5.19 getCode

```
app3.apis.getCode(address [, defaultBlock] [, callback])
```

Get the code at a specific address.

5.19.1 Parameters

1. `String` - The address to get the code from.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *app3.apis.defaultBlock*.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.19.2 Returns

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

5.19.3 Example

```
app3.apis.getCode("0xd5677cf67b5aa051bb40496e68ad359eb97cfbf8")
.then(console.log);
>
↪"0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060078202905091900
↪"
```

5.20 getBlock

```
app3.apis.getBlock(blockHashOrBlockNumber [, returnTransactionObjects] [, callback])
```

Returns a block matching the block number or block hash.

5.20.1 Parameters

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

5.20.2 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` - `Number`: Balance of miner (10 blocks ago).
- `txTrieHash` 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.
- `coinbase` - `String`: The address of the beneficiary to whom the mining rewards were given.
- `coinbaseMask` - `String`: The mask of the coinbase.
- `rewardPoint` - `String`: Integer of the RP for this block of miner.
- `cumulativeRewardPoint` - `String`: Integer of the cumulative RP of the chain until this block.
- `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.
- `mineralUsed` - `Number`: The total used mineral 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.
- `logsBloom` 256 Bytes - `String`: The bloom filter for the logs of the block. `null` when its pending block.
- `mnHash` 32 Bytes - `String`: Hash of the masternodes
- `mnReward` - `Number`: Base amount of Masternode rewards
- `mnGenerals` - `Array`: Array of general masternodes.
- `mnMajors` - `Array`: Array of major masternodes.
- `mnPrivates` - `Array`: Array of private masternodes.
- `size` - `Number`: Integer the size of this block in bytes.

5.20.3 Example

```
app3.apis.getBlock(3150)
.then(console.log);

> { number: 3000,
    hash: '0xb3b51d689be882447e2a9a94be94c67603a921f9394af013a2ce0b4657f3f93d',
    parentHash: '0x62c51711e287f3330e8efe638c7b166788f732ad758eaebc4707720125b02ff6',
    coinbase: '0x643d122906cdaa4468702696431da16dec7d5ad2',
    stateRoot: '0xbca98d3790068316689df10bbc0851d32df79c23400fa93a36121ff04a44b1a4',
    txTrieHash: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
    receiptsTrieHash:
↪'0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
    rewardPoint: '682438316916457871243767603736587500000000',
    cumulativeRewardPoint: '651838904459052714671924990415770050647282427640800000',
    gasLimit: 50000000,
    gasUsed: 0,
    mineralUsed: '0',
    timestamp: 1541696090,
    extraData: '0x4150495320706f776572656420536572766572',
    rpSeed: '0x356e54d588078c71cf1737ec360accc1c5e8a4a94bf77eb4429421587b5f0fa8',
    nonce: '0x3bcbc228e38b6ce5f100',
    transactions: [],
    logsBloom:
↪'0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪',
    mnHash: '',
    mnReward: '0.000000000000000000',
    mnGenerals: [],
    mnMajors: [],
    mnPrivates: [],
    size: 613 }
```

5.21 getBlockTransactionCount

```
app3.apis.getBlockTransactionCount(blockHashOrBlockNumber [, callback])
```

Returns the number of transaction in a given block.

5.21.1 Parameters

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

5.21.2 Returns

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

5.21.3 Example

```
app3.apis.getBlockTransactionCount(
↪'0xb3b51d689be882447e2a9a94be94c67603a921f9394af013a2ce0b4657f3f93d')
.then(console.log);
> 0
```

5.22 getTransaction

```
app3.apis.getTransaction(transactionHash [, callback])
```

Returns a transaction matching the given transaction hash.

5.22.1 Parameters

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

5.22.2 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.
- `toMask` - `String`: Mask of the receiver address. `null` when its a contract creation transaction or not registered.
- `value` - `String`: Value transferred in wei.
- `valueAPIS` - `String`: Value readable.
- `gasPrice` - `String`: Gas price provided by the sender in wei.
- `gasPriceAPIS` - `String`: Gas price readable.
- `gas` - `Number`: Gas provided by the sender.
- `feeLimitAPIS` - `String`: The maximum chargeable amount (gasLimit x gasPrice).
- `data` - `String`: The data sent along with the transaction.
- `r` - `String`: ‘r’ value of signature. ‘‘null‘ when its not signed.

- `s` - `String`: 's' value of signature. ''null' when its not signed.
- `v` - `String`: 'v' value of signature. ''null' when its not signed.
- `certR` - `String`: 'r' value of authentication. `null` when its not authorized by knowledge key.
- `certS` - `String`: 's' value of authentication. `null` when its not authorized by knowledge key.
- `certV` - `String`: 'v' value of authentication. `null` when its not authorized by knowledge key.

5.22.3 Example

```
app3.apis.getTransaction(
↪'0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b$234')
.then(console.log);

> { hash: '0x960d6a7bfa4db0b32b59c473af05d9ae6975d0b45f29d38b83564d26cc9342dc',
    nonce: 4,
    blockHash: '0x9dd1ec4ca42d54805e8dbdbd37909a9dfd91a783d18dd7cbb9bbe3805ec777b3',
    blockNumber: 40995,
    transactionIndex: 0,
    from: '0xea31B942F886fcBBcFeDd5580F992Afe464A38B8',
    to: '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe',
    value: '10000000000000000000000',
    valueAPIS: '10_000',
    gas: 200000,
    gasPrice: '50000000000',
    gasPriceAPIS: '0.00000005',
    feeLimitAPIS: '0.01',
    data: '',
    r: '0x4b01bc76bc2682d04e9a7cdcb3526d56da54e63d1a7087f0607f3e0f72744e66',
    s: '0x6628fda2e57ebc99a3e59acef5e95f7063d9ff06d1fa76d33a84c5032d576d44',
    v: '0x1c' }
```

5.23 getTransactionsByKeyword

```
app3.apis.getTransactionsByKeyword(keyword, rowCount, offset)
```

Returns a transaction list matching the given keyword.

5.23.1 Parameters

1. `String` - Keywords for retrieving transactions. ie. transaction hash, address, address mask.
2. `Number` - Maximum number of search results.
3. `Number` - Number of skipped search results.

5.23.2 Returns

`Promise` returns `Object` - A transaction object its hash `transactionHash`:

- `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.
- `timestamp` - `Number`: The unix timestamp for when the block was collated.
- `transactionHash` 32 Bytes - `String`: Hash of the transaction.
- `from` - `String`: Address of the sender.
- `to` - `String`: Address of the receiver. `null` when its a contract creation transaction.
- `toMask` - `String`: Mask of the receiver. `null` when its not registered.
- `contractAddress` - `String`: The contract address created, if the transaction was a contract creation, otherwise `null`.
- `gas` - `Number`: Gas provided by the sender.
- `gasPrice` - `String`: Gas price provided by the sender in atto.
- `gasPriceAPIS` - `String`: Gas price in APIS.
- `gasUsed`- `Number`: The amount of gas used by this specific transaction alone.
- `mineralUsed` - `String`: The amount of mineral used by this specific transaction alone.
- `mineralUsedMNR` - `String`: Used mineral in MNR.
- `feePaid` - `String`: Finally paid fee amount in atto.
- `feePaidAPIS` - `String`: Paid fee amount in APIS.

5.23.3 Example

```
  app3.apis.getTransactionsByKeyword('apis@me', 10, 0)
  .then(console.log);

  > [ { status: '0x01',
  transactionHash: '0x7a14dbb3b0bf4d2fcb03c4ddf9dbb503ee1a4342c39bf702484a48c8d6898335
↪',
  blockHash: '0xc53da41b4e58211307709654a96caff537fb99d6f2827487da76d96b6886825e',
  blockNumber: 323,
  timestamp: '1545202180',
  from: '0x891122cb40b2a83b2686107720d84c7eb7f37ad4',
  to: '0x1000000000000000000000000000000000037453',
  gas: 266581,
  gasPrice: '50000000000',
  gasPriceAPIS: '0.00000005',
  gasUsed: 265971,
  fee: '13298550000000000',
  feeAPIS: '0.01329855',
  mineralUsed: '13298550000000000',
  mineralUsedMNR: '0.01329855',
  feePaid: '0',
  feePaidAPIS: '0' },
{ status: '0x01',
  transactionHash: '0xaffe6524f8af6484d23bdd362709e3c26373999175a72a356cc6174636814293
↪',
  blockHash: '0xf591aef448e402998b87fdc6237be2e02a3b7721118a40468afcea216de6924a',
  blockNumber: 318,
  timestamp: '1545202140',
```

(continues on next page)

(continued from previous page)

```
  from: '0x891122cb40b2a83b2686107720d84c7eb7f37ad4',
  to: '0x1000000000000000000000000000000000037453',
  gas: 266581,
  gasPrice: '50000000000',
  gasPriceAPIS: '0.00000005',
  gasUsed: 265971,
  fee: '13298550000000000',
  feeAPIS: '0.01329855',
  mineralUsed: '13298550000000000',
  mineralUsedMNR: '0.01329855',
  feePaid: '0',
  feePaidAPIS: '0' } ]
```

5.24 getRecentTransactions

```
app3.apis.getRecentTransactions([rowCount] [, offset])
```

Returns a recent transaction list matching the given condition.

5.24.1 Parameters

1. `Number` - Maximum number of search results. Default is 20
2. `Number` - Number of skipped search results. Default is 0

5.24.2 Returns

`Promise` returns `Object` - A list of transaction object:

- `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.
- `timestamp` - `Number`: The unix timestamp for when the block was collated.
- `transactionHash` 32 Bytes - `String`: Hash of the transaction.
- `from` - `String`: Address of the sender.
- `to` - `String`: Address of the receiver. `null` when its a contract creation transaction.
- `toMask` - `String`: Mask of the receiver. `null` when its not registered.
- `contractAddress` - `String`: The contract address created, if the transaction was a contract creation, otherwise `null`.
- `gas` - `Number`: Gas provided by the sender.
- `gasPrice` - `String`: Gas price provided by the sender in atto.
- `gasPriceAPIS` - `String`: Gas price in APIS.
- `gasUsed`- `Number`: The amount of gas used by this specific transaction alone.

- `mineralUsed` - `String`: The amount of mineral used by this specific transaction alone.
- `mineralUsedMNR` - `String`: Used mineral in MNR.
- `feePaid` - `String`: Finally paid fee amount in atto.
- `feePaidAPIS` - `String`: Paid fee amount in APIS.

5.24.3 Example

```
  app3.apis.getRecentTransactions(0, 0)
  .then(console.log);

  > [ { status: '0x01',
  transactionHash: '0x0172b3308acd1149b2100f20434752ba3978a64ae8c26231b5e994f7e18cba2e
↪',
  blockHash: '0xab3bae560a30289894bdb809e342ee4292df0e2ecdc379d852f06f0e3994f4e4',
  blockNumber: 1049,
  timestamp: '1545293545',
  from: '0x89112261d222abc94d257acb4ad470ca911fcfb6',
  to: '0x89112261d222abc94d257acb4ad470ca911fcfb6',
  gas: 220000,
  gasPrice: '77777777777',
  gasPriceAPIZ: '0.000000077777777777',
  gasUsed: 213000,
  fee: '16566666666501000',
  feeAPIZ: '0.016566666666501',
  mineralUsed: '16566666666501000',
  mineralUsedMNR: '0.016566666666501',
  feePaid: '0',
  feePaidAPIZ: '0' },
{ status: '0x01',
  transactionHash: '0x744b4e976c63a68f168044b4268be011918ae43a28b77b37515af4443a2d1fbd
↪',
  blockHash: '0x59798b2ee3c7c6dab018c4379337927f6f0838dcf85ca365f0d78cbaf6bdb832',
  blockNumber: 1047,
  timestamp: '1545293529',
  from: '0x89112261d222abc94d257acb4ad470ca911fcfb6',
  to: '0x891122cb40b2a83b2686107720d84c7eb7f37ad4',
  gas: 200000,
  gasPrice: '50000000000',
  gasPriceAPIZ: '0.00000005',
  gasUsed: 200000,
  fee: '10000000000000000',
  feeAPIZ: '0.01',
  mineralUsed: '10000000000000000',
  mineralUsedMNR: '0.01',
  feePaid: '0',
  feePaidAPIZ: '0' }, ... ]
```

5.25 getRecentBlocks

```
app3.apis.getRecentBlock([rowCount] [, offset])
```

Returns a block list of recent confirmed.

5.25.1 Parameters

1. `Number` - Maximum number of search results. Default is 20
2. `Number` - Number of skipped search results. Default is 1

5.25.2 Returns

`Promise` returns `Array` - The list of 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` - `Number`: Balance of miner (10 blocks ago).
- `txTrieHash` 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.
- `coinbase` - `String`: The address of the beneficiary to whom the mining rewards were given.
- `coinbaseMask` - `String`: The mask of the coinbase.
- `rewardPoint` - `String`: Integer of the RP for this block of miner.
- `cumulativeRewardPoint` - `String`: Integer of the cumulative RP of the chain until this block.
- `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.
- `mineralUsed` - `Number`: The total used mineral 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.
- `logsBloom` 256 Bytes - `String`: The bloom filter for the logs of the block. `null` when its pending block.
- `mnHash` 32 Bytes - `String`: Hash of the masternodes
- `mnReward` - `Number`: Base amount of Masternode rewards
- `mnGenerals` - `Array`: Array of general masternodes.
- `mnMajors` - `Array`: Array of major masternodes.
- `mnPrivates` - `Array`: Array of private masternodes.
- `size` - `Number`: Integer the size of this block in bytes.

5.25.3 Example

```
app3.apis.getRecentBlocks()
.then(console.log);

> [ { number: 1196,
```

(continues on next page)

(continued from previous page)

```
  hash: '0x5c6eb8638c7636bd033636e0cea62d667ccc4224a552f93390e12b021c68e822',
  parentHash: '0x2280ee46a61af1557d1562f7c241ec31a51694c713772a1176d51565d655e820',
  coinbase: '0xc779d025076d31d26d4e73af90df1f298ebe5b0a',
  stateRoot: '0xa72bc8f9d5a0fd1696a43d4745ca1146cd4a28252c85af3d48d99c1546a33392',
  txTrieHash: '0x83633cadfa85ff78c477e90c9b804364a9c9bdb0d4c362e189ef3464819edafb',
  receiptsTrieHash:
↪'0x771dbd196da19ef46e74b7d7e57444bfbcf48ebbd3eae20903849d9219a1fcaa',
  rewardPoint: '10183221216695777122220329400000000000000000',
  cumulativeRewardPoint: '4270753430417833658920855884570051589608925547500000000',
  gasLimit: 200000000,
  gasUsed: 200000,
  mineralUsed: '1800000000000000',
  timestamp: '1545294721',
  extraData: '0x4150495320706f77657265642053657276657272',
  rpSeed: '0xfac3cce593d5c93c514baf22c6fea42f002bb1ec2eb24f593d7553174111202a',
  nonce: '0x01e6b2c7688471080000',
  txSize: 1,
  transactions:
   [ '0xed514ce278bf40160a71b85c2023c850a493fdb6880cad1118434d1c057f99af' ],
  logsBloom:
↪'0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪',
  size: 743 },
{ number: 1197,
  hash: '0x0804718c1f632054efc8fc00e88ffaa978b07e2195ea333923f9edca1fdabd60',
  parentHash: '0x5c6eb8638c7636bd033636e0cea62d667ccc4224a552f93390e12b021c68e822',
  coinbase: '0x643d122906cdaa4468702696431da16dec7d5ad2',
  stateRoot: '0xeb23f13319b1ffdbd8df80054311afc66c50a72ac7f9c8fcf1ac0577028a9192',
  txTrieHash: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
  receiptsTrieHash:
↪'0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
  rewardPoint: '17947445068020020579741482505926999650000000000000',
  cumulativeRewardPoint: '4270755225162340460922913858718302182308890547500000000',
  gasLimit: 200000000,
  gasUsed: 0,
  mineralUsed: '0',
  timestamp: '1545294729',
  extraData: '0x4150495320706f77657265642053657276657272',
  rpSeed: '0xb64c17597776982dd4e2bbc493aa660ae0b6cc2dd8203252657b789d7816e419',
  nonce: '0x2e178224f7d0f953a800',
  txSize: 0,
  transactions: [],
  size: 616 }, ... ]
```

5.26 getTransactionFromBlock

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

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

5.26.1 Parameters

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

5.26.2 Returns

`Promise` returns `Object` - A transaction object, see *app3.apis.getTransaction*:

5.26.3 Example

```
var transaction = app3.apis.getTransactionFromBlock('0x4534534534', 2)
.then(console.log);
> // see app3.apis.getTransaction
```

5.27 getTransactionReceipt

```
app3.apis.getTransactionReceipt(hash [, callback])
```

Returns the receipt of a transaction by transaction hash.

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

5.27.1 Parameters

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

5.27.2 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.
- `timestamp`- `Number`: The unix timestamp for when the block was collated.
- `from` - `String`: Address of the sender.

- `to` - `String`: Address of the receiver. `null` when its a contract creation transaction.
- `toMask` - `String`: Mask of the receiver. `null` when its not registered.
- `contractAddress` - `String`: The contract address created, if the transaction was a contract creation, otherwise `null`.
- `gas` - `Number`: Gas provided by the sender.
- `gasPrice` - `String`: Gas price provided by the sender in atto.
- `gasPriceAPIS` - `String`: Gas price in APIS.
- `gasUsed`- `Number`: The amount of gas used by this specific transaction alone.
- `mineralUsed` - `String`: The amount of mineral used by this specific transaction alone.
- `mineralUsedMNR` - `String`: Used mineral in MNR.
- `feePaid` - `String`: Finally paid fee amount in atto.
- `feePaidAPIS` - `String`: Paid fee amount in APIS.
- `cumulativeGasUsed` - `Number`: The total amount of gas used when this transaction was executed in the block.
- `cumulativeMineralUsed` - `Number`: The total amount of mineral used when this transaction was executed in the block.
- `cumulativeMineralUsedMNR` - `Number`: Cumulative mineral in MNR
- `logs` - `Array`: Array of log objects, which this transaction generated.
- `internalTransaction` - `Array`: Array of internal transaction objects.

5.27.3 Example

```
var receipt = app3.apis.getTransactionReceipt(
↪'0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b')
.then(console.log);

> { status: true,
    transactionHash:
↪'0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b',
    transactionIndex: 0,
    blockHash: '0x9dd1ec4ca42d54805e8dbdbd37909a9dfd91a783d18dd7cbb9bbe3805ec777b3',
    blockNumber: 40995,
    from: '0xea31b942f886fcbbcfedd5580f992afe464a38b8',
    to: '0xea31b942f886fcbbcfedd5580f992afe464a38b8',
    gas: 200000,
    gasPrice: '50000000000',
    gasPriceAPIS: '0.00000005',
    gasUsed: 200000,
    fee: '10000000000000000',
    feeAPIS: '0.01',
    mineralUsed: '10503000000000',
    mineralUsedMNR: '0.000010503',
    feePaid: '9989497000000000',
    feePaidAPIS: '0.009989497',
    cumulativeGasUsed: 200000,
    cumulativeMineralUsed: '10503000000000',
```

(continues on next page)

(continued from previous page)

```
    cumulativeMineralUsedMNR: '0.000010503'
    logs: [{
         // logs as returned by getPastLogs, etc.
    }, ...] }
```

5.28 getTransactionCount

```
app3.apis.getTransactionCount(address [, defaultBlock] [, callback])
```

Get the numbers of transactions sent from this address.

5.28.1 Parameters

1. `String` - The address to get the numbers of transactions from.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *app3.apis.defaultBlock*.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

5.28.2 Returns

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

5.28.3 Example

```
app3.apis.getTransactionCount("0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe")
.then(console.log);
> 1
```

5.29 getMasternodeCount

```
app3.apis.getMasternodeCount([, callback])
```

Get the number of masternodes.

Masternode State :

- Earlybird - Masternode that can be joined through apis.mn. The nodes are joined to the day(10,800 blocks) before the round begins.
- Normal - Masternode joined through APIS Core within the first day(10,800 blocks) of the round
- Late - Masternode joined through the APIS Core after the normal participation period of the round

5.29.1 Parameters

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

5.29.2 Returns

`Promise` returns `Object` - The number of masternodes.

- `generalEarly` - `Number`: Number of General Earlybird Masternodes
- `majorEarly` - `Number`: Number of Major Earlybird Masternodes
- `privateEarly` - `Number`: Number of Private Earlybird Masternodes
- `generalNormal` - `Number`: Number of General Normal Masternodes
- `majorNormal` - `Number`: Number of Major Normal Masternodes
- `privateNormal` - `Number`: Number of Private Normal Masternodes
- `generalLate` - `Number`: Number of General Late Masternodes
- `majorLate` - `Number`: Number of Major Late Masternodes
- `privateLate` - `Number`: Number of Private Late Masternodes

5.29.3 Example

```
app3.apis.getMasternodeCount()
.then(console.log);
> {
    generalEarly: 3929,
    majorEarly: 2661,
    privateEarly: 1776,
    generalNormal: 0,
    majorNormal: 0,
    privateNormal: 0,
    generalLate: 2,
    majorLate: 1,
    privateLate: 3 }
```

5.30 sendTransaction

```
app3.apis.sendTransaction(transactionObject [, callback])
```

Sends a transaction to the network.

5.30.1 Parameters

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

- `from` - `String|Number`: The address for the sending account. Uses the app3.apis.defaultAccount property, if not specified. Or an address or index of a local wallet in app3.apis.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 atto, 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 atto, defaults to *app3.apis.gasPrice*.
- `data` - `String`: (optional) Either a ABI byte string containing the data of the function call on a contract, or in the case of a contract-creation transaction the initialisation code.
- `nonce` - `Number`: (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

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

Note: The `from` property can also be an address or index from the app3.apis.accounts.wallet. It will then sign locally using the private key of that account, and send the transaction via *app3.apis.sendSignedTransaction()*.

5.30.2 Returns

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

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

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

5.30.3 Example

```
// compiled solidity source code using https://remix.ethereum.org
var code =
↪"603d80600c6000396000f3007c0100000000000000000000000000000000000000000000000000000060003504
↪";


// using the callback
app3.apis.sendTransaction({
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    data: code // deploying a contracrt
}, function(error, hash){
```

(continues on next page)

(continued from previous page)

```
    ...
});

// using the promise
app3.apis.sendTransaction({
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    to: '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe',
    value: '1000000000000000'
})
.then(function(receipt){
    ...
});


// using the event emitter
app3.apis.sendTransaction({
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    to: '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe',
    value: '1000000000000000'
})
.on('transactionHash', function(hash){
    ...
})
.on('receipt', function(receipt){
    ...
})
.on('confirmation', function(confirmationNumber, receipt){ ... })
.on('error', console.error); // If a out of gas error, the second parameter is the␣
↪receipt.
```

5.31 sendSignedTransaction

```
app3.apis.sendSignedTransaction(signedTransactionData [, callback])
```

Sends an already signed transaction, generated for example using *app3.apis.accounts.signTransaction*.

5.31.1 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.

5.31.2 Returns

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

Please see the return values for *app3.apis.sendTransaction* for details.

5.31.3 Example

```
const rawTx =
↪'0xf889808609184e72a000822710940000000000000000000000000000000000000000000080a47f74657374320000000000000
↪';

app3.apis.sendSignedTransaction(rawTx)
.on('receipt', console.log);

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

5.32 sign

```
app3.apis.sign(dataToSign, address [, callback])
```

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

5.32.1 Parameters

1. `String` - Data to sign. If String it will be converted using app3.utils.utf8ToHex.
2. `String|Number` - Address to sign data with. Or an address or index of a local wallet in app3.apis.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 app3.apis.accounts.wallet. It will then sign locally using the private key of this account.

5.32.2 Returns

`Promise` returns `String` - The signature.

5.32.3 Example

```
app3.apis.sign("Hello world", "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪"

// the below is the same
app3.apis.sign(app3.utils.utf8ToHex("Hello world"),
↪"0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪"
```

5.33 call

```
app3.apis.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.

5.33.1 Parameters

1. `Object` - A transaction object see *app3.apis.sendTransaction*, with the difference that for calls the `from` property is optional as well.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *app3.apis.defaultBlock*.
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 returned data of the call, e.g. a smart contract functions return value.

5.33.3 Example

```
app3.apis.call({
    to: "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe", // contract address
    data: "0xc6888fa1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000003"
})
.then(console.log);
> "0x000000000000000000000000000000000000000000000000000000000000000a"
```

5.34 estimateGas

```
app3.apis.estimateGas(callObject [, callback])
```

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

5.34.1 Parameters

1. `Object` - A transaction object see *app3.apis.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.

5.34.2 Returns

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

5.34.3 Example

```
app3.apis.estimateGas({
    to: "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
    data: "0xc6888fa1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000003"
})
.then(console.log);
> "0x0000000000000000000000000000000000000000000000000000000000000015"
```

5.35 getPastLogs

```
app3.apis.getPastLogs(options [, callback])
```

Gets past logs, matching the given options.

5.35.1 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']]`

5.35.2 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.

5.35.3 Example

```
app3.apis.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'
},{...}]
```

CHAPTER 6

app3.apis.subscribe

The `app3.apis.subscribe` function lets you subscribe to specific events in the blockchain.

6.1 subscribe

```
app3.apis.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.

6.1.3 Notification returns

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

6.1.4 Example

```
var subscription = app3.apis.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

```
app3.apis.clearSubscriptions()
```

Resets subscriptions.

6.2.1 Parameters

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

6.2.2 Returns

`Boolean`

6.2.3 Example

```
app3.apis.subscribe('logs', {} ,function(){ ... });

...

app3.apis.clearSubscriptions();
```

6.3 subscribe("pendingTransactions")

```
app3.apis.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 = app3.apis.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")

```
app3.apis.subscribe('newBlockHeaders' [, callback]);
```

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

6.4.1 Parameters

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

6.4.2 Returns

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

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

The structure of a returned block header is as follows:

- `number` - `Number`: The block number. `null` when its pending block.
- `hash` 32 Bytes - `String`: Hash of the block. `null` when its pending block.
- `parentHash` 32 Bytes - `String`: Hash of the parent block.
- `coinbase` - `String`: The address of the beneficiary to whom the mining rewards were given.
- `stateRoot` 32 Bytes - `String`: The root of the final state trie of the block.
- `txTrieHash` 32 Bytes - `String`: The root of the transaction trie of the block
- `receiptsTrieHash` 32 Bytes - `String`: The root of the receipts.
- `rewardPoint` 32 Bytes - `Number`: RewardPoint of coinbase for proof-of-stake
- `cumulativeRewardPoint` 32 Bytes - `Number`: RewardPoint accumulated up to the current block
- `gasLimit` - `Number`: The maximum gas allowed in this block.
- `gasUsed` - `Number`: The total used gas by all transactions in this block.
- `mineralUsed` - `Number`: The total used mineral by all transactions in this block.
- `timestamp` - `Number`: The unix timestamp for when the block was collated.
- `extraData` - `String`: The "extra data" field of this block.
- `logsBloom` 256 Bytes - `String`: The bloom filter for the logs of the block. `null` when its pending block.
- `rpSeed` 32 Bytes - `String`: Hash of the generated proof-of-stake. `null` when its pending block.
- `nonce` 32 Bytes - `String`: Balance of coinbase for proof-of-stake. `null` when its pending block.

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 = app3.apis.subscribe('newBlockHeaders', function(error, result){
    if (!error) {
        console.log(result);

        return;
    }

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

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

6.5 subscribe("logs")

```
app3.apis.subscribe('logs', options [, callback]);
```

Subscribes to incoming logs, filtered by the given options.

6.5.1 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.

6.5.2 Returns

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

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

For the structure of a returned event `Object` see *app3.apis.getPastEvents return values*.

6.5.3 Notification returns

1. `Object|Null` - First parameter is an error object if the subscription failed.
2. `Object` - The log object like in *app3.apis.getPastEvents return values*.

6.5.4 Example

```
var subscription = app3.apis.subscribe('logs', {
    address: '0x123456..',
    topics: ['0x12345...']
}, function(error, result){
    if (!error)
        console.log(result);
})
.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

app3.apis.Contract

The `app3.apis.Contract` object makes it easy to interact with smart contracts on the APIS blockchain. When you create a new contract object you give it the json interface of the respective smart contract and app3 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 app3.apis.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, can be added later using `myContract.options.address = '0x1234..'`
3. **`options` - `Object` (optional): The options of the contract. Some are used as fallbacks for calls and transactions:**

    - `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 app3.apis.Contract([...],
↪'0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe', {
    from: '0x1234567890123456789012345678901234567891', // default from address
    gasPrice: '50000000000' // default gas price in aAPIS, 50 nAPIS in this case
});
```

7.2 = Properties =

7.3 options

```
myContract.options
```

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

7.3.1 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).

7.3.2 Example

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

(continues on next page)

(continued from previous page)

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

7.4 options.address

```
myContract.options.address
```

The address used for this contract instance. All transactions generated by app3js from this contract will contain this address as the "to".

The address will be stored in lowercase.

7.4.1 Property

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

7.4.2 Example

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

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

7.5 options.jsonInterface

```
myContract.options.jsonInterface
```

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

7.5.1 Property

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

7.5.2 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}],
}]

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

7.6 = Methods =

7.7 clone

```
myContract.clone()
```

Clones the current contract instance.

7.7.1 Parameters

none

7.7.2 Returns

`Object`: The new contract instance.

7.7.3 Example

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

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

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

7.8 deploy

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

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

7.8.1 Parameters

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

7.8.2 Returns

`Object`: The transaction object:

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

  For details to the methods see the documentation below.

7.8.3 Example

```
myContract.deploy({
    data: '0x12345...',
    arguments: [123, 'My String']
})
.send({
    from: '0x1234567890123456789012345678901234567891',
    gas: 1500000,
    gasPrice: '30000000000000'
}, 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...';
```

(continues on next page)

(continued from previous page)

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

7.9 methods

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

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

The methods of this smart contract are available through:

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

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

7.9.1 Parameters

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

7.9.2 Returns

`Object`: The transaction object:

- `Array` - arguments: The arguments passed to the method before. They can be changed.
- `Function` - *call*: Will call the "constant" method and execute its smart contract method in the EVM without sending a transaction (Can't alter the smart contract state).
- `Function` - *send*: Will send a transaction to the smart contract and execute its method (Can alter the smart contract state).
- `Function` - *estimateGas*: Will estimate the gas used when the method would be executed on chain.
- `Function` - *encodeABI*: Encodes the ABI for this method. This can be send using a transaction, call the method or passing into another smart contracts method as argument.

  For details to the methods see the documentation below.

7.9.3 Example

```
// calling a method

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

7.10 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 can not alter the smart contract state.

7.10.1 Parameters

1. **`options` - `Object` (optional): The options used for calling.**
    - `from` - `String` (optional): The address the call "transaction" should be made from.
    - `gasPrice` - `String` (optional): The gas price in aAPIS 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.

7.10.2 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:

7.10.3 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!%");
    }
}

// app3js
var MyContract = new app3.apis.Contract(abi, address);
MyContract.methods.myFunction().call()
.then(console.log);
> Result {
    myNumber: '23456',
```

(continues on next page)

(continued from previous page)

```
    myString: 'Hello!%',
    0: '23456', // these are here as fallbacks if the name is not know or given
    1: 'Hello!%'
}


// SINGLE-ARGUMENT RETURN:

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

// app3js
var MyContract = new app3.apis.Contract(abi, address);
MyContract.methods.myFunction().call()
.then(console.log);
> "Hello!%"
```

7.11 methods.myMethod.send

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

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

7.11.1 Parameters

1. **`options` - `Object`: The options used for sending.**
    - `from` - `String`: The address the transaction should be sent from.
    - `gasPrice` - `String` (optional): The gas price in aAPIS 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 aAPIS.
2. `callback` - `Function` (optional): This callback will be fired first with the “transactionHash”, or with an error object as the first argument.

7.11.2 Returns

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

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

- `"transactionHash"` returns `String`: is fired right after the transaction is sent and a transaction hash is available.

- `"receipt"` returns `Object`: is fired when the transaction *receipt* is available. Receipts from contracts will have no `logs` property, but instead an `events` property with event names as keys and events as properties. See *getPastEvents return values* for details about the returned event object.
- `"confirmation"` returns `Number`, `Object`: is fired for every confirmation up to the 24th confirmation. Receives the confirmation number as the first and the receipt as the second argument. Fired from confirmation 1 on, which is the block where it's minded.
- `"error"` returns `Error`: is fired if an error occurs during sending. If a out of gas error, the second parameter is the receipt.

7.11.3 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'
                },
```

(continues on next page)

(continued from previous page)

```
                raw: {
                    data:
↪'0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385',
                    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', console.error); // If there's an out of gas error the second parameter␣
↪is the receipt.
```

7.12 methods.myMethod.estimateGas

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

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

7.12.1 Parameters

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

7.12.2 Returns

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

7.12.3 Example

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

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

7.13 methods.myMethod.encodeABI

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

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

7.13.1 Parameters

none

7.13.2 Returns

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

7.13.3 Example

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

7.14 = Events =

7.15 once

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

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

7.15.1 Parameters

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

7.15.2 Returns

`undefined`

7.15.3 Example

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

7.16 events

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

Subscribe to an event

7.16.1 Parameters

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

7.16.2 Returns

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

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

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

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

- `blockHash` 32 Bytes - `String`: Hash of the block this event was created in. `null` when it's still pending.
- `blockNumber` - `Number`: The block number this log was created in. `null` when still pending.
- `raw.data` - `String`: The data containing non-indexed log parameter.
- `raw.topics` - `Array`: An array with max 4 32 Byte topics, topic 1-3 contains indexed parameters of the event.

7.16.3 Example

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

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

7.17 events.allEvents

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

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

7.18 getPastEvents

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

Gets past events for this contract.

7.18.1 Parameters

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

7.18.2 Returns

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

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

7.18.3 Example

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

(continues on next page)

(continued from previous page)

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

CHAPTER 8

app3.apis.accounts

The `app3.apis.accounts` contains functions to generate APIS 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('app3-apis-accounts');

// Passing in the apis or app3 package is necessary to allow retrieving chainId,␣
↪gasPrice and nonce automatically
// for accounts.signTransaction().
var accounts = new Accounts('ws://localhost:8546');
```

8.1 create

```
app3.apis.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 *app3.apis.accounts.signTransaction()* for more.
- `sign(data)` - `Function`: The function to sign transactions. See *app3.apis.accounts.sign()* for more.

8.1.3 Example

```
app3.apis.accounts.create();
> {
    address: "0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01",
    privateKey: "0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}

app3.apis.accounts.create('2435@#@#@±±±±!!!!
↪67854321345676432 1§34567543213456785432134567');
> {
    address: "0xF2CD2AA0c7926743B1D4310b2BC984a0a453c3d4",
    privateKey: "0xd7325de5c2c1cf0009fac77d3d04a9c004b038883446b065871bc3e831dcd098",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}

app3.apis.accounts.create(app3.utils.randomHex(32));
> {
    address: "0xe78150FaCD36E8EB00291e251424a0515AA1FF05",
    privateKey: "0xcc505ee6067fba3f6fc2050643379e190e087aeffe5d958ab9f2f3ed3800fa4e",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}
```

8.2 privateKeyToAccount

```
app3.apis.accounts.privateKeyToAccount(privateKey);
```

Creates an account object from a private key.

8.2.1 Parameters

1. `privateKey` - `String`: The private key to convert.

8.2.2 Returns

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

8.2.3 Example

```
app3.apis.accounts.privateKeyToAccount(
↪'0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709');
> {
    address: '0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01',
    privateKey: '0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709',
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}

app3.apis.accounts.privateKeyToAccount(
↪'0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709');
> {
    address: '0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01',
    privateKey: '0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709',
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}
```

8.3 signTransaction

```
app3.apis.accounts.signTransaction(tx, privateKey [, callback]);
```

Signs an APIS 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 *app3.apis.getTransactionCount()*.
    - `chainId` - `String`: (optional) The chain id to use when signing this transaction. Default will use app3.apis.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 *app3.apis.gasPrice()*
    - `gas` - `String`: The gas provided by the transaction.
2. `privateKey` - `String`: The private key to sign with.

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

8.3.2 Returns

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

- `messageHash` - `String`: The hash of the given message.
- `r` - `String`: First 32 bytes of the signature
- `s` - `String`: Next 32 bytes of the signature
- `v` - `String`: Recovery value + 27
- `rawTransaction` - `String`: The RLP encoded transaction, ready to be send using *app3.apis.sendSignedTransaction*.

8.3.3 Example

```
app3.apis.accounts.signTransaction({
    to: '0xF0109fC8DF283027b6285cc889F5aA624EaC1F55',
    value: '1000000000',
    gas: 2000000
}, '0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318')
.then(console.log);
> {
    messageHash: '0x88cfbd7e51c7a40540b233cf68b62ad1df3e92462f1c6018d6d67eae0f3b08f5',
    v: '0x25',
    r: '0xc9cf86333bcb065d140032ecaab5d9281bde80f21b9687b3e94161de42d51895',
    s: '0x727a108a0b8d101465414033c3f705a9c7b826e596766046ee1183dbc8aeaa68',
    rawTransaction:
↪'0xf869808504e3b29200831e848094f0109fc8df283027b6285cc889f5aa624eac1f55843b9aca008025a0c9cf86333bcb
↪'
}

app3.apis.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
↪'
}
```

8.4 authTransaction

```
app3.apis.accounts.authTransaction(tx, privateKey, knowledgeKey [, callback]);
```

Authorize an APIS transaction with a given knowledge key.

8.4.1 Parameters

1. **`tx` - `Object`: The transaction object as follows:**
    - `nonce` - `String`: (optional) The nonce to use when signing this transaction. Default will use *app3.apis.getTransactionCount()*.
    - `chainId` - `String`: (optional) The chain id to use when signing this transaction. Default will use app3.apis.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 *app3.apis.gasPrice()*
    - `gas` - `String`: The gas provided by the transaction.
2. `privateKey` - `String`: The private key to sign with.
3. `knowledgeKey` - `String`: The knowledge key to authorization with.
4. `callback` - `Function`: (optional) Optional callback, returns an error object as first parameter and the result as second.

8.4.2 Returns

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

- `messageHash` - `String`: The hash of the given message.
- `sigR` - `String`: First 32 bytes of the signature
- `sigS` - `String`: Next 32 bytes of the signature
- `sigV` - `String`: Recovery value + 27
- `certR` - `String`: First 32 bytes of the certificate
- `certS` - `String`: Next 32 bytes of the certificate
- `certV` - `String`: Recovery value + 27
- `rawTransaction` - `String`: The RLP encoded transaction, ready to be send using *app3.apis.sendSignedTransaction*.

8.4.3 Example

```
app3.apis.accounts.authTransaction({
    to: '0xF0109fC8DF283027b6285cc889F5aA624EaC1F55',
    value: '1000000000',
    gas: 2000000
}, '0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318', 'Test!@1234')
.then(console.log);
> {
    messageHash: '0xbdb7a230bff3aac997f725408ecc7b691a7e2bfec7fda1c96089e6a54707b1f0',
    sigV: '0x25',
    sigR: '0xc0894df696594d2a8aa0a6d6621e189e90c2cb014fd43d7cfcaa1a2b47994011',
    sigS: '0x256414c62457b62a27b21cab13a8f97c56576deb1b66d730e24d73c2dd418c00',
    certV: '0x25',
    certR: '0x176328849514d3c24c60db889cc450c2b86f099beb3c2984ebefb8b0e382818c',
    certS: '0x1aa1b872c6cb3946637c272f8bf81439f33cea3155d2ce8d65451d8db859e659',
    rawTransaction:
↪'0xf8ad80850ba43b7400831e848094f0109fc8df283027b6285cc889f5aa624eac1f5580843b9aca008025a0c0894df696
↪'
  }
```

8.5 recoverTransaction

```
app3.apis.accounts.recoverTransaction(rawTransaction);
```

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

8.5.1 Parameters

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

8.5.2 Returns

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

8.5.3 Example

```
app3.apis.accounts.recoverTransaction(
↪'0xf86180808401ef364594f0109fc8df283027b6285cc889f5aa624eac1f5580801ca031573280d608f75137e33fc14655
↪');
> "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55"
```

8.6 hashMessage

```
app3.apis.accounts.hashMessage(message);
```

Hashes the given message to be passed *app3.apis.accounts.recover()* function. The data will be UTF-8 HEX decoded and enveloped as follows: `"\x19APIS Signed Message:\n" + message.length + message` and hashed using keccak256.

8.6.1 Parameters

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

8.6.2 Returns

`String`: The hashed message

8.6.3 Example

```
app3.apis.accounts.hashMessage("Hello World")
> "0xa1de988600a42c4b4ab089b619297c17d53cffae5d5120d82d8a92d0bb3b78f2"

// the below results in the same hash
app3.apis.accounts.hashMessage(app3.utils.utf8ToHex("Hello World"))
> "0xa1de988600a42c4b4ab089b619297c17d53cffae5d5120d82d8a92d0bb3b78f2"
```

8.7 sign

```
app3.apis.accounts.sign(data, privateKey);
```

Signs arbitrary data. This data is before UTF-8 HEX decoded and enveloped as follows: `"\x19APIS Signed Message:\n" + message.length + message`.

8.7.1 Parameters

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

8.7.2 Returns

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

- `message` - `String`: The the given message.
- `messageHash` - `String`: The hash of the given message.
- `r` - `String`: First 32 bytes of the signature
- `s` - `String`: Next 32 bytes of the signature

- `v` - `String`: Recovery value + 27

8.7.3 Example

```
app3.apis.accounts.sign('Some data',
↪'0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318');
> {
    message: 'Some data',
    messageHash: '0x1da44b586eb0729ff70a73c326926f6ed5a25f5b056e7f47fbc6e58d86871655',
    v: '0x1c',
    r: '0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd',
    s: '0x6007e74cd82e037b800186422fc2da167c747ef045e5d18a5f5d4300f8e1a029',
    signature:
↪'0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd6007e74cd82e037b800186422fc2da16
↪'
}
```

8.8 recover

```
app3.apis.accounts.recover(signatureObject);
app3.apis.accounts.recover(message, signature [, preFixed]);
app3.apis.accounts.recover(message, v, r, s [, preFixed]);
```

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

8.8.1 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 `"\x19APIS 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 `"\x19APIS Signed Message:\n" + message.length + message`, and assumed to be already prefixed.

8.8.2 Returns

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

8.8.3 Example

```
app3.apis.accounts.recover({
    messageHash: '0x1da44b586eb0729ff70a73c326926f6ed5a25f5b056e7f47fbc6e58d86871655',
    v: '0x1c',
    r: '0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd',
    s: '0x6007e74cd82e037b800186422fc2da167c747ef045e5d18a5f5d4300f8e1a029'
})
> "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23"

// message, signature
app3.apis.accounts.recover('Some data',
↪'0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd6007e74cd82e037b800186422fc2da1
↪');
> "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23"

// message, v, r, s
app3.apis.accounts.recover('Some data', '0x1c',
↪'0xb91467e570a6466aa9e9876cbcd013baba02900b8979d43fe208a4a4f339f5fd',
↪'0x6007e74cd82e037b800186422fc2da167c747ef045e5d18a5f5d4300f8e1a029');
> "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23"
```

8.9 encrypt

```
app3.apis.accounts.encrypt(privateKey, password);
```

Encrypts a private key to the app3 keystore v3 standard.

8.9.1 Parameters

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

8.9.2 Returns

`Object`: The encrypted keystore v3 JSON.

8.9.3 Example

```
app3.apis.accounts.encrypt(
↪'0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318', 'test!')
> {
    version: 3,
    id: '04e9bcbb-96fa-497b-94d1-14df4cd20af6',
    address: '2c7536e3605d9c16a7a3d7b1898e529396a65c23',
    crypto: {
        ciphertext: 'a1c25da3ecde4e6a24f3697251dd15d6208520efc84ad97397e906e6df24d251
↪',
```

(continues on next page)

(continued from previous page)

```
        cipherparams: { iv: '2885df2b63f7ef247d753c82fa20038a' },
        cipher: 'aes-128-ctr',
        kdf: 'scrypt',
        kdfparams: {
            dklen: 32,
            salt: '4531b3c174cc3ff32a6a7a85d6761b410db674807b2d216d022318ceee50be10',
            n: 262144,
            r: 8,
            p: 1
        },
        mac: 'b8b010fff37f9ae5559a352a185e86f9b9c1d7f7a9f1bd4e82a5dd35468fc7f6'
    }
}
```

8.10 decrypt

```
app3.apis.accounts.decrypt(keystoreJsonV3, password);
```

Decrypts a keystore v3 JSON, and creates the account.

8.10.1 Parameters

1. `encryptedPrivateKey` - `String`: The encrypted private key to decrypt.
2. `password` - `String`: The password used for encryption.

8.10.2 Returns

`Object`: The decrypted account.

8.10.3 Example

```
app3.apis.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
```

(continues on next page)

(continued from previous page)

```
        },
        mac: 'b8b010fff37f9ae5559a352a185e86f9b9c1d7f7a9f1bd4e82a5dd35468fc7f6'
    }
}, 'test!');
> {
    address: "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23",
    privateKey: "0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}
```

8.11 wallet

```
app3.apis.accounts.wallet;
```

Contains an in memory wallet with multiple accounts. These accounts can be used when using *app3.apis.sendTransaction()*.

8.11.1 Example

```
app3.apis.accounts.wallet;
> Wallet {
    0: {...}, // account by index
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...},  // same account by address
    "0xf0109fc8df283027b6285cc889f5aa624eac1f55": {...},  // same account by address␣
↪lowercase
    1: {...},
    "0xD0122fC8DF283027b6285cc889F5aA624EaC1d23": {...},
    "0xd0122fc8df283027b6285cc889f5aa624eac1d23": {...},

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

    length: 2,
}
```

8.12 wallet.create

```
app3.apis.accounts.wallet.create(numberOfAccounts [, entropy]);
```

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

8.12.1 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.

8.12.2 Returns

`Object`: The wallet object.

8.12.3 Example

```
app3.apis.accounts.wallet.create(2,
↪'54674321§3456764321§345674321§3453647544±±±§±±±!!!43534534534534');
> Wallet {
    0: {...},
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...},
    "0xf0109fc8df283027b6285cc889f5aa624eac1f55": {...},
    ...
}
```

8.13 wallet.add

```
app3.apis.accounts.wallet.add(account);
```

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

8.13.1 Parameters

1. `account` - `String|Object`: A private key or account object created with app3.apis.accounts.create().

8.13.2 Returns

`Object`: The added account.

8.13.3 Example

```
app3.apis.accounts.wallet.add(
↪'0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318');
> {
    index: 0,
    address: '0x2c7536E3605D9C16a7a3D7b1898e529396a65c23',
    privateKey: '0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318',
    signTransaction: function(tx){...},
    sign: function(data){...},
```

(continues on next page)

(continued from previous page)

```
    encrypt: function(password){...}
}

app3.apis.accounts.wallet.add({
    privateKey: '0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709',
    address: '0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01'
});
> {
    index: 0,
    address: '0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01',
    privateKey: '0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709',
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}
}
```

8.14 wallet.remove

```
app3.apis.accounts.wallet.remove(account);
```

Removes an account from the wallet.

8.14.1 Parameters

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

8.14.2 Returns

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

8.14.3 Example

```
app3.apis.accounts.wallet;
> Wallet {
    0: {...},
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...}
    1: {...},
    "0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01": {...}
    ...
}

app3.apis.accounts.wallet.remove('0xF0109fC8DF283027b6285cc889F5aA624EaC1F55');
> true

app3.apis.accounts.wallet.remove(3);
> false
```

8.15 wallet.clear

```
app3.apis.accounts.wallet.clear();
```

Securely empties the wallet and removes all its accounts.

8.15.1 Parameters

none

8.15.2 Returns

`Object`: The wallet object.

8.15.3 Example

```
app3.apis.accounts.wallet.clear();
> Wallet {
    add: function(){},
    remove: function(){},
    save: function(){},
    load: function(){},
    clear: function(){},

    length: 0
}
```

8.16 wallet.encrypt

```
app3.apis.accounts.wallet.encrypt(password);
```

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

8.16.1 Parameters

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

8.16.2 Returns

`Array`: The encrypted keystore v3.

8.16.3 Example

```
app3.apis.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' } }
]
```

8.17 wallet.decrypt

```
app3.apis.accounts.wallet.decrypt(keystoreArray, password);
```

Decrypts keystore v3 objects.

8.17.1 Parameters

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

8.17.2 Returns

`Object`: The wallet object.

8.17.3 Example

```
app3.apis.accounts.wallet.decrypt([
  { version: 3,
  id: '83191a81-aaca-451f-b63d-0c5f3b849289',
  address: '06f702337909c06c82b09b7a22f0a2f0855d1f68',
  crypto:
   { ciphertext: '7d34deae112841fba86e3e6cf08f5398dda323a8e4d29332621534e2c4069e8d',
     cipherparams: { iv: '497f4d26997a84d570778eae874b2333' },
     cipher: 'aes-128-ctr',
```

(continues on next page)

(continued from previous page)

```
    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": {...}
    ...
}
```

8.18 wallet.save

```
app3.apis.accounts.wallet.save(password [, keyName]);
```

Stores the wallet encrypted and as string in local storage.

Note: Browser only.

8.18.1 Parameters

1. `password` - `String`: The password to encrypt the wallet.
2. `keyName` - `String`: (optional) The key used for the local storage position, defaults to `"app3js_wallet"`.

8.18.2 Returns

`Boolean`

8.18.3 Example

```
app3.apis.accounts.wallet.save('test#!$');
> true
```

8.19 wallet.load

```
app3.apis.accounts.wallet.load(password [, keyName]);
```

Loads a wallet from local storage and decrypts it.

Note: Browser only.

8.19.1 Parameters

1. `password` - `String`: The password to decrypt the wallet.
2. `keyName` - `String`: (optional) The key used for the localstorage position, defaults to `"app3js_wallet"`.

8.19.2 Returns

`Object`: The wallet object.

8.19.3 Example

```
app3.apis.accounts.wallet.load('test#!$', 'myWalletKey');
> Wallet {
    0: {...},
    1: {...},
    "0xF0109fC8DF283027b6285cc889F5aA624EaC1F55": {...},
    "0xD0122fC8DF283027b6285cc889F5aA624EaC1d23": {...}
    ...
}
```

CHAPTER 9

app3.apis.personal

The `app3-apis-personal` package allows you to interact with the APIS 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('app3-apis-personal');

var personal = new Personal('ws://some.local-or-remote.node:8546');


// or using the app3 umbrella package

var App3 = require('app3');
var app3 = new App3('ws://some.local-or-remote.node:8546');

// -> app3.apis.personal
```

9.1 newAccount

```
app3.apis.personal.newAccount(password, [callback])
```

Create a new account on the node that App3js is connected to with its provider. The RPC method used is `personal_newAccount`. It differs from app3.apis.accounts.create() where the key pair is created only on client and it's up to the developer to manage it.

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

9.1.1 Parameters

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

9.1.2 Returns

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

9.1.3 Example

```
app3.apis.personal.newAccount('!@superpassword')
.then(console.log);
> '0x1234567891011121314151617181920212223456'
```

9.2 sign

```
app3.apis.personal.sign(dataToSign, address, password [, callback])
```

Signs data using a specific account.

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

9.2.1 Parameters

1. `String` - Data to sign. If String it will be converted using app3.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.

9.2.2 Returns

`Promise` returns `String` - The signature.

9.2.3 Example

```
app3.apis.personal.sign("Hello world", "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe",
↪"test password!")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪"
```

(continues on next page)

(continued from previous page)

```
// the below is the same
app3.apis.personal.sign(app3.utils.utf8ToHex("Hello world"),
↪"0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe", "test password!")
.then(console.log);
>
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪"
```

9.3 ecRecover

```
app3.apis.personal.ecRecover(dataThatWasSigned, signature [, callback])
```

Recovers the account that signed the data.

9.3.1 Parameters

1. `String` - Data that was signed. If String it will be converted using app3.utils.utf8ToHex.
2. `String` - The signature.
3. `Function` - (optional) Optional callback, returns an error object as first parameter and the result as second.

9.3.2 Returns

`Promise` returns `String` - The account.

9.3.3 Example

```
app3.apis.personal.ecRecover("Hello world",
↪"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dc
↪").then(console.log);
> "0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe"
```

9.4 signTransaction

```
app3.apis.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.

9.4.1 Parameters

1. `Object` - The transaction data to sign *app3.apis.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.

9.4.2 Returns

`Promise` returns `Object` - The RLP encoded transaction. The `raw` property can be used to send the transaction using *app3.apis.sendSignedTransaction*.

9.4.3 Example

```
app3.apis.personal.signTransaction({
    from: "0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01",
    gasPrice: "50000000000",
    gas: "200000",
    to: '0x3535353535353535353535353535353535353535',
    value: "1000000000000000000",
    data: ""
}, 'MyPassword!').then(console.log);
> {
    raw:
↪'0xf87180850ba43b740083030d4094353535353535353535353535353535353535353535353580880de0b6b3a76400008025a009
↪',
    tx: {
        hash: '0x185da901f30171b573d06ffdf1cbe8aa7e624eb3d207e4c6cede5c28c0fd948c',
        nonce: '0x',
        from: '0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01',
        to: '0x3535353535353535353535353535353535353535',
        value: '1000000000000000000',
        valueAPIS: '1',
        gas: '200000',
        gasPrice: '50000000000',
        gasPriceAPIS: '0.00000005',
        feePaidAPIS: '0.01',
        data: '',
        r: '0x0975caffb4cb5c6f9dfe97315d3c448926ee0d3be85a44c1006be525d38ef5a3',
        s: '0x1b8c0d9ac65615823dbdc0dfd398f5807d0582f490b2b408224e4803d46c3b69',
        v: '0x1b'
    }
}
```

9.5 sendTransaction

```
app3.apis.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.

9.5.1 Parameters

1. `Object` - The transaction options
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.

9.5.2 Returns

`Promise` returns `string` - The transaction hash

9.5.3 Example

```
app3.apis.personal.sendTransaction({
    from: "0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01",
    gasPrice: "50000000000",
    gas: "200000",
    to: '0x3535353535353535353535353535353535353535',
    value: "1000000000000000000",
    data: ""
}, 'MyPassword!').then(console.log);
> '0x185da901f30171b573d06ffdf1cbe8aa7e624eb3d207e4c6cede5c28c0fd948c'
```

9.6 unlockAccount

```
app3.apis.personal.unlockAccount(address, password, unlockDuraction [, callback])
```

Unlocks the given account.

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

9.6.1 Parameters

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

9.6.2 Returns

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

9.6.3 Example

```
app3.apis.personal.unlockAccount("0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe", "test␣
↪password!", 600)
.then(console.log('Account unlocked!'));
> "Account unlocked!"
```

9.7 lockAccount

```
app3.apis.personal.lockAccount(address [, callback])
```

Locks the given account.

9.7.1 Parameters

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

9.7.2 Returns

`Promise` returns `boolean` - True if the account got locked successful otherwise false.

9.7.3 Example

```
app3.apis.personal.lockAccount("0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe")
.then(console.log('Account locked!'));
> "Account locked!"
```

9.8 getAccounts

```
app3.apis.personal.getAccounts([callback])
```

Returns a list of accounts the node controls by using the provider and calling the RPC method `personal_listAccounts`. Using app3.apis.accounts.create() will not add accounts into this list. For that use app3.apis.personal.newAccount().

The results are the same as `app3.apis.getAccounts()` except that calls the RPC method `apis_accounts`.

9.8.1 Returns

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

9.8.2 Example

```
app3.apis.personal.getAccounts()
.then(console.log);
> [ {
        address: 'ceffabff83caf9594d12e101e93c71a3aaea96ef',
        index: '0',
        aAPIS: '0',
        aMNR: '28008000000000',
        nonce: '2',
        APIS: '0',
        MNR: '0.000028008'
    },
    {
        address: 'b8ce9ab6943e0eced004cde8e3bbed6568b2fa01',
        index: '1',
        aAPIS: '5000000000000000000000',
        aMNR: '20000000000000000',
        nonce: '0',
        APIS: '5,000',
        MNR: '0.02'
    } ]
```

9.9 importRawKey

```
app3.apis.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.

9.9.1 Parameters

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

9.9.2 Returns

`Promise` returns `String` - The address of the account.

9.9.3 Example

```
app3.apis.personal.importRawKey(
↪"cd3376bb711cb332ee3fb2ca04c6a8b9f70c316fcdf7a1f44ef4c7999483295e", "password1234")
.then(console.log);
> "0x8f337bf484b2fc75e4b0436645dcc226ee2ac531"
```

CHAPTER 10

app3.apis.abi

The `app3.apis.abi` functions let you de- and encode parameters to ABI (Application Binary Interface) for function calls.

10.1 encodeFunctionSignature

```
app3.apis.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.

10.1.1 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)`

10.1.2 Returns

`String` - The ABI signature of the function.

10.1.3 Example

```
// From a JSON interface object
app3.apis.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
app3.apis.abi.encodeFunctionSignature('myMethod(uint256,string)')
> '0x24ee0097'
```

10.2 encodeEventSignature

```
app3.apis.abi.encodeEventSignature(eventName);
```

Encodes the event name to its ABI signature, which are the sha3 hash of the event name including input types.

10.2.1 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)`

10.2.2 Returns

`String` - The ABI signature of the event.

10.2.3 Example

```
app3.apis.abi.encodeEventSignature('myEvent(uint256,bytes32)')
> 0xf2eeb729e636a8cb783be044acf6b7b1e2c5863735b60d6daae84c366ee87d97

// or from a json interface object
app3.apis.abi.encodeEventSignature({
    name: 'myEvent',
    type: 'event',
    inputs: [{
        type: 'uint256',
        name: 'myNumber'
    },{
        type: 'bytes32',
        name: 'myBytes'
    }]
})
> 0xf2eeb729e636a8cb783be044acf6b7b1e2c5863735b60d6daae84c366ee87d97
```

10.3 encodeParameter

```
app3.apis.abi.encodeParameter(type, parameter);
```

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

10.3.1 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.

10.3.2 Returns

`String` - The ABI encoded parameter.

10.3.3 Example

```
app3.apis.abi.encodeParameter('uint256', '2345675643');
> "0x000000000000000000000000000000000000000000000000000000008bd02b7b"

app3.apis.abi.encodeParameter('uint256', '2345675643');
> "0x000000000000000000000000000000000000000000000000000000008bd02b7b"

app3.apis.abi.encodeParameter('bytes32', '0xdf3234');
> "0xdf3234000000000000000000000000000000000000000000000000000000000000"

app3.apis.abi.encodeParameter('bytes', '0xdf3234');
>
↪"0x0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000
↪"

app3.apis.abi.encodeParameter('bytes32[]', ['0xdf3234', '0xfdfd']);
>
↪"0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000
↪"

app3.apis.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
↪"
```

10.4 encodeParameters

```
app3.apis.abi.encodeParameters(typesArray, parameters);
```

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

10.4.1 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.

10.4.2 Returns

`String` - The ABI encoded parameters.

10.4.3 Example

```
app3.apis.abi.encodeParameters(['uint256','string'], ['2345675643', 'Hello!%']);
>
↪"0x0000000000000000000000000000000000000000000000000000000008bd02b7b0000000000000000000000000000000000
↪"

app3.apis.abi.encodeParameters(['uint8[]','bytes32'], [['34','434'], '0x324567fff']);
> "0x0

app3.apis.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
↪"
```

10.5 encodeFunctionCall

```
app3.apis.abi.encodeFunctionCall(jsonInterface, parameters);
```

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

10.5.1 Parameters

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

10.5.2 Returns

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

10.5.3 Example

```
app3.apis.abi.encodeFunctionCall({
    name: 'myMethod',
    type: 'function',
    inputs: [{
        type: 'uint256',
        name: 'myNumber'
    },{
        type: 'string',
        name: 'myString'
    }]
}, ['2345675643', 'Hello!%']);
>
↪"0x24ee00970000000000000000000000000000000000000000000000000000000008bd02b7b0000000000000000000000000
↪"
```

10.6 decodeParameter

```
app3.apis.abi.decodeParameter(type, hexString);
```

Decodes an ABI encoded parameter to its JavaScript type.

10.6.1 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.

10.6.2 Returns

`Mixed` - The decoded parameter.

10.6.3 Example

```
app3.apis.abi.decodeParameter('uint256',
↪'0x0000000000000000000000000000000000000000000000000000000000000010');
> "16"

app3.apis.abi.decodeParameter('string',
↪'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪');
> "Hello!%!"

app3.apis.abi.decodeParameter('string',
↪'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
↪');
> "Hello!%!"

app3.apis.abi.decodeParameter(
    {
        "ParentStruct": {
          "propertyOne": 'uint256',
          "propertyTwo": 'uint256',
          "childStruct": {
            "propertyOne": 'uint256',
            "propertyTwo": 'uint256'
          }
        }
    },

,
↪'0x0000000000000000000000000000000000000000000000000000000000000002a0000000000000000000000000000000000
↪');
> {
    '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'
    }
}
```

10.7 decodeParameters

```
app3.apis.abi.decodeParameters(typesArray, hexString);
```

Decodes ABI encoded parameters to its JavaScript types.

10.7.1 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.

10.7.2 Returns

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

10.7.3 Example

```
app3.apis.abi.decodeParameters(['string', 'uint256'],
↪'0x0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000
↪');
> Result { '0': 'Hello!%!', '1': '234' }

app3.apis.abi.decodeParameters([{
    type: 'string',
    name: 'myString'
},{
    type: 'uint256',
    name: 'myNumber'
}],
↪'0x0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000
↪');
> Result {
    '0': 'Hello!%!',
    '1': '234',
    myString: 'Hello!%!',
    myNumber: '234'
}

app3.apis.abi.decodeParameters([
  'uint8[]',
  {
    "ParentStruct": {
      "propertyOne": 'uint256',
      "propertyTwo": 'uint256',
      "childStruct": {
        "propertyOne": 'uint256',
        "propertyTwo": 'uint256'
      }
    }
  }
],
↪'0x0000000000000000000000000000000000000000000000000000000000000060000000000000000000000000000000000000
↪');
> 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'
    }
}
```

10.8 decodeLog

```
app3.apis.abi.decodeLog(inputs, hexString, topics);
```

Decodes ABI encoded log data and indexed topic data.

10.8.1 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].

10.8.2 Returns

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

10.8.3 Example

```
app3.apis.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 11

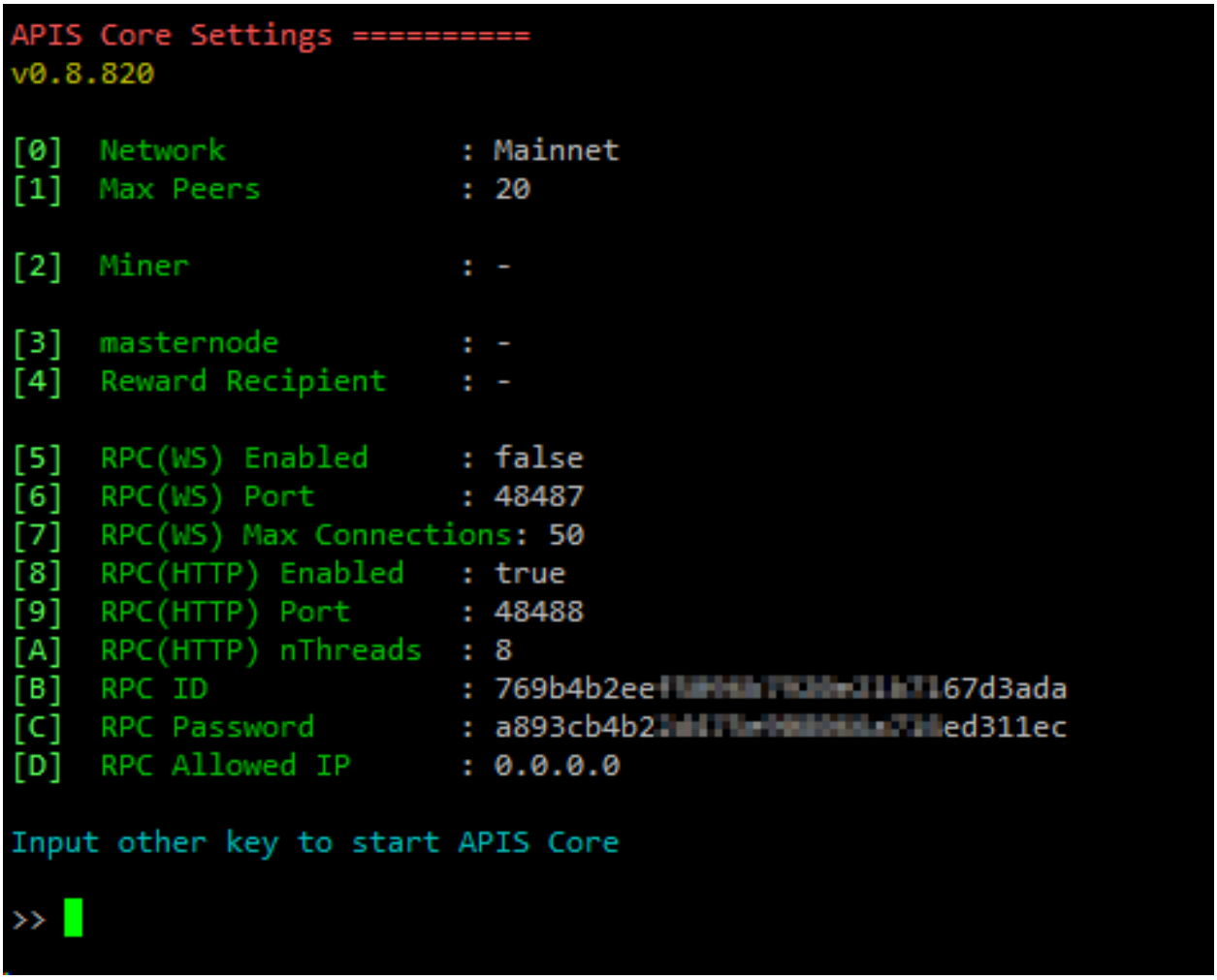
API

11.1 Setting up APIS Core HTTP RPC

You can run the HTTP RPC server through the APIS Core program under Linux.

For APIS Core installation instructions, please refer to the following documentation

When you run APIS Core, the following configuration options are displayed on the screen.



To activate the HTTP RPC server, press '8' on your keyboard and press 'Enter'.

When the setting of "RPC (HTTP) Enabled" item is changed to true, use of the HTTP RPC server is activated.

To access the server, keep the port number of [9], ID of [B], and Password of [C].

If you need to connect from external, set the value of [D] item to 0.0.0.0 or the IP address you want to connect.

Finally, press 'Enter' key to start the APIS Core program without entering anything.

Once the block synchronization starts, you can access the RPC server.

11.1.1 Authenticating using a ID and Password

As additional protection for your request traffic, you should use HTTP Basic Authentication to access API.

```
curl --user RPC-ID:RPC-PASSWORD http://127.0.0.1:45678
```

11.2 Securing Your Credentials

11.2.1 Authenticating using a ID and Password

As additional protection for your request traffic, you should use HTTP Basic Authentication to access API.

```
curl --user RPC-ID:RPC-PASSWORD http://127.0.0.1:45678
```

11.3 Make Requests

Below is a quick command line example using *curl*:

```
$ curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_blockNumber", "params": []}' \
    --user RPC-ID:RPC-Password \
    http://127.0.0.1:45678
```

The result should look something like this :

```
$ {"id":1,"jsonrpc":"2.0","method":"apis_blockNumber","result":"0x00000000001ea7eb"}
```

Note: "0x00000000001ea7eb" will be replaced with the block number of the most recent block on that network

---

11.4 apis_protocolVersion

Returns the APIS protocol version of the node

11.4.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_protocolVersion", "params":␣
↪[]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.4.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_protocolVersion",
  "result": "2.0"
}
```

11.5 apis_syncing

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

11.5.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_syncing", "params": []}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.5.2 RESPONSE

`result` returns `Object|Boolean` - A sync object when the node is currently syncing or `false`:

- `startingBlock` - `Hex`: The block number where the sync started.
- `currentBlock` - `Hex`: The block number where at which block the node currently synced to already.
- `highestBlock` - `Hex`: The estimated block number to sync to.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_syncing",
  "result": {
    "startingBlock": "0x001E81C0",
    "currentBlock": "0x001EA894",
    "highestBlock": "0x001EA894"
  }
}
```

11.6 apis_coinbase

Returns the coinbase address to which mining rewards will go.

11.6.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_coinbase", "params": []}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.6.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_coinbase",
  "result": "0xbaf1abfc83c7f9594d12e101e93c71a3a6ec9fe9"
}
```

11.7 apis_mining

Checks whapiser the node is mining or not.

11.7.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_mining", "params": []}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.7.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_mining",
  "result": true
}
```

11.8 apis_gasPrice

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

11.8.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_gasPrice", "params": []}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.8.2 RESPONSE

`result` returns `String` - Hex string of the current gas price in wei.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_gasPrice",
  "result": "0x0000000ba43b7400"
}
```

11.9 apis_accounts

Returns a list of accounts the node controls.

11.9.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_accounts", "params": []}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.9.2 RESPONSE

`result` returns `Array` - Array of accounts controlled by node objects.

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

- `address` 32 Bytes - `String`: address of account
- `index` - `Number`: The index position of accounts controlled by node.
- `aAPIS` - `Number`: The current APIS balance for the given account in atto.
- `aMNR` - `Number`: The current MNR balance for the given account in atto.
- `nonce` - `Number`: The number of transactions
- `APIS` 32 Bytes - `String`: The current APIS balance for the given account.
- `MNR` 32 Bytes - `String`: The current MNR balance for the given account.
- `proofKey` 32 Bytes - `String`: 2-Step Verification Key. `null` if not registered.
- `isMasternode` = `Boolean`: True if given address is a masternode.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_accounts",
  "result": [
    {
      "address": "0xbaf1abfc83c7f9594d12e101e93c71a3a6ec9fe9",
      "index": "0",
```

(continues on next page)

(continued from previous page)

```
      "aAPIS": "28008000000000",
      "aMNR": "0",
      "nonce": "2",
      "APIS": "0.000028008",
      "MNR": "0"
    }
  ]
}
```

11.10 apis_getWalletInfo

Returns a information of given address.

11.10.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_getWalletInfo", "params": [
↪"c5f590c1035ae780906514ff8e76dd86b89b97dc"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678

curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_getWalletInfo", "params": [
↪"some_name@me"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.10.2 RESPONSE

`result` returns `Object` - Information of given address.

- `address` 32 Bytes - `String`: address of account.
- `mask` - `String`: Mask of given address. `null` if not registered.
- `aAPIS` - `Number`: The current APIS balance for the given account in atto.
- `aMNR` - `Number`: The current MNR balance for the given account in atto.
- `nonce` - `Number`: The number of transactions
- `APIS` 32 Bytes - `String`: The current APIS balance for the given account.
- `MNR` 32 Bytes - `String`: The current MNR balance for the given account.
- `proofKey` 32 Bytes - `String`: 2-Step Verification Key. `null` if not registered.
- `isContract` - `String`: True if given address has contract code. `null` if hasn't.
- `isMasternode` = `Boolean`: True if given address is a masternode.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getWalletInfo",
  "result": {
    "address": "f481dcace2750647fdad784b8981c3c16c1412cf",
    "mask": "some_name@me",
    "aAPIS": "0",
    "aMNR": "6480000000000000000",
    "nonce": "8",
    "APIS": "0",
    "MNR": "6.48",
    "Reward": "6,211.7274536893",
    "aReward": "6211727453689300000000"
  }
}
```

11.11 apis_blockNumber

Returns the current block number.

11.11.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_blockNumber", "params": []}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.11.2 RESPONSE

`result` returns `String` - Hex string of the most recent block.


```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_blockNumber",
  "result": "0x00000000001eac3e"
}
```

11.12 apis_getBalance

Get the balance of an address.

11.12.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_getBalance", "params": [
↪"0xb29efff525b229b6efbe37979467e7c64b8a84ce","latest"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.12.2 RESPONSE

`result` returns `Object` - The current balance for the given address in atto.

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

- `aAPIS` - `Number`: The current APIS balance for the given account in atto.
- `aMNR` - `Number`: The current MNR balance for the given account in atto.
- `APIS` - `Number`: The current readable APIS balance for the given account.
- `MNR` - `Number`: The current readable MNR balance for the given account.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getBalance",
  "result": {
    "aAPIS": "132488647894811551500000000",
    "aMNR": "926670950000000000",
    "APIS": "13,248,864.78948115515",
    "MNR": "0.92667095"
  }
}
```

11.13 apis_getCode

Get the code at a specific address.

11.13.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc": "2.0", "id": 1, "method": "apis_getCode", "params": [
↪"866962b19d403a712f2c6bca390f9f295ba2dfe9"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.13.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getCode",
  "result":
↪"0x6080604052600436106102515760003557c0100000000000000000000000000000000000000000000000000.
↪.."
}
```

11.14 apis_getBlockByNumber

Returns a block matching the block number.

11.14.1 REQUEST

1. `Number` - The block number.
2. `Boolean` - (optional, default `false`) If `true`, the returned block will contain all transactions as objects, if `false` it will only contains the transaction hashes.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getBlockByNumber","params":["0x2710
↪",false]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.14.2 RESPONSE

`result` 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` - `Number`: Balance of miner (10 blocks ago).
- `txTrieHash` 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.
- `coinbase` - `String`: The address of the beneficiary to whom the mining rewards were given.
- `coinbaseMask` - `String`: The mask of the coinbase.
- `rewardPoint` - `String`: Integer of the RP for this block of miner.
- `cumulativeRewardPoint` - `String`: Integer of the cumulative RP of the chain until this block.
- `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.

- `mineralUsed` - `Number`: The total used mineral 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.
- `logsBloom` 256 Bytes - `String`: The bloom filter for the logs of the block. `null` when its pending block.
- `mnHash` 32 Bytes - `String`: Hash of the masternodes
- `mnReward` - `Number`: Base amount of Masternode rewards
- `mnGenerals` - `Array`: Array of general masternodes.
- `mnMajors` - `Array`: Array of major masternodes.
- `mnPrivates` - `Array`: Array of private masternodes.
- `size` - `Number`: Integer the size of this block in bytes.


```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getBlockByNumber",
  "result": {
    "number": 10000,
    "hash": "0x93c601f19b570fb689eceb6b1551b8cb7e8b6d1f6cfe980fd139496cedcd7a7e",
    "parentHash": "0xa89ad9435bcec17c3f8e399ec47afc7f98888179fd14abc54ea53df3c22d2833
↪",
    "coinbase": "0x66a99a95246aa66237514f8aa03e2386351cf432",
    "coinbaseMask": "",
    "stateRoot": "0xdb9980ea807e0a7196f763cf52617b893a49f0529f7463f745ab3517b3dcfef3",
    "txTrieHash": "0xf1c97cbdc42aa950cdd248efecd218d58ca04455616e4134ef9a6cbe6e7b7c4a
↪",
    "receiptsTrieHash":
↪"0xffd51add8233eb9185f25c65f11b58c88035f39029c7fb1bbc95aa0d43a2198c",
    "rewardPoint": "111479185684725693459940247439624104312000000000",
    "cumulativeRewardPoint": "85742440859928735227161577432932586091332801000000000",
    "gasLimit": 100000000,
    "gasUsed": 10347216,
    "mineralUsed": "10000000000000",
    "timestamp": "1546141681",
    "extraData": "0x4150495320706f776572656420536572766572",
    "rpSeed": "0x8af9dea17f02f394d3cd947f9f53e2e7c0a1fa348555a0a92aec9a4305b3046a",
    "nonce": "0x258dcd45652bd3d69000",
    "txSize": 11,
    "transactions": [
      "0x3235deae37b6aba20a56d1958394314803a84fab4d861f09677767bc2329a897",
      "0x5949bca2a8e9819834e85e14253d189e7e94d142880f9ebef086d3220d071d1e",
      "0xc9aa5ec867fbb47e44c75947bb5e4e57805f0810c9ad733302f52d548287aa8d",
      "0xcea711fe5000d2cd185757df6e8aa7b4f8d3858203d2d3aa97870b0e453f26e4",
      "0x9409ce9e169ff6007cbcaa3137d98bb385d3de1911d135544330cab903e863de",
      "0xd2006fc207f496c0f405744145caa82bf1f6fb44076c4f93583cc0029392baed",
      "0x76b02d08372af92f097f50b476d7984ea20da22a59af78e30fc85653a46e0dbb",
      "0x9c22b73a09cd05098c71faea4b688aa6b583d350b8a3a5c283377229e4bc9f5c",
      "0x084c5f07e5d4dc2fd101e275acdf71640d855bea7ea3ceeaf49aaab0797359db",
      "0x690a1a22262d18a3a0968bb367c270aa2ee5511917768f65eb6ac7e29e7b4bc2",
      "0x2f43ba31c6bbea3ecd08ad528344d1930bd82786cdff54b0c0df503f14a6987e"
    ],
    "logsBloom":
↪"0x00000000000020000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
```


(continues on next page)

(continued from previous page)

```
    "size": 2695
  }
}
```

11.15 apis_getBlockTransactionCountByNumber

Returns the number of transaction in a given block number.

11.15.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getBlockTransactionCountByNumber",
↪"params":["0x2710"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.15.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getBlockTransactionCountByNumber",
  "result": "0x0000000b"
}
```

11.16 apis_getTransactionByHash

Returns a transaction matching the given transaction hash.

11.16.1 REQUEST

1. `String` - The transaction hash.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getTransactionByHash","params":[
↪"3235deae37b6aba20a56d1958394314803a84fab4d861f09677767bc2329a897"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.16.2 RESPONSE

`result` returns `Object` - A transaction object, see *app3.apis.getTransaction*:


```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getTransactionByHash",
  "result": {
    "hash": "0x3235deae37b6aba20a56d1958394314803a84fab4d861f09677767bc2329a897",
    "nonce": "0x11d1",
    "blockHash": "0x93c601f19b570fb689eceb6b1551b8cb7e8b6d1f6cfe980fd139496cedcd7a7e",
    "blockNumber": "10000",
    "transactionIndex": "0x0000000000000000",
    "timestamp": "1546141681",
    "from": "0xc5f590c1035ae780906514ff8e76dd86b89b97dc",
    "to": "0x866962b19d403a712f2c6bca390f9f295ba2dfe9",
    "value": "2000000000000000000000000",
    "valueAPIS": "200,000",
    "gas": "4000000",
    "gasPrice": "50000000000",
    "gasPriceAPIS": "0.00000005",
    "feePaidAPIS": "0.2",
    "data":
↪"0x2c32ac980000000000000000000000000c5f590c1035ae780906514ff8e76dd86b89b97dc0000000000000000000000000
↪",
    "r": "0x62c934e772f9487e969464341f276ac83ee042d09f1ed569fe44aa78e3be0be2",
    "s": "0x38eff978bd0cc4b1116001422d0153546383f2966daa278ee11e9aade90e05ed",
    "v": "0x1c"
  }
}
```

11.17 apis_getTransactionsByKeyword

Returns a transaction list matching the given keyword.

11.17.1 REQUEST

1. `String` - Keywords for retrieving transactions. ie. transaction hash, address, address mask.
2. `String` - Hex string of search results.
3. `String` - Hex string of skipped search results.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getTransactionsByKeyword","params
↪":["platform@!","0x01","0x0"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.17.2 RESPONSE

`result` returns `Array` - The list of transaction object, see : see *app3.apis.getTransaction*:

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getTransactionsByKeyword",
  "result": [
    {
      "status": "0x01",
      "transactionHash":
↪"0xa6f58a7bfbb01c5bf36459b3822942e38a43fbdbc4c77a85b6fdb15e96a38aa8",
      "transactionIndex": 0,
      "blockHash": "0xac3eadc390d85c66b27a90ecdc6450a177d32c9302f13c3bcadecf258493",
      "blockNumber": 815509,
      "timestamp": "1553437790",
      "from": "0xf7f52b29fb123c56e79429b6a7e8797c64ce4b8a",
      "fromMask": "test@me",
      "to": "0x16bd780f5aff9a4f8eff003945fbd0251f5cf203",
      "value": "123075600000000000000000",
      "valueAPIS": "1,230,756",
      "nonce": 5,
      "gas": 200000,
      "gasPrice": "50000000000",
      "gasPriceAPIS": "0.00000005",
      "gasUsed": 200000,
      "fee": "10000000000000000",
      "feeAPIS": "0.01",
      "mineralUsed": "10000000000000000",
      "mineralUsedMNR": "0.01",
      "feePaid": "0",
      "feePaidAPIS": "0",
      "cumulativeGasUsed": 200000,
      "cumulativeMineralUsed": "10000000000000000",
      "cumulativeMineralUsedMNR": "0.01",
      "data": ""
    }
  ]
}
```


11.18 apis_getRecentTransactions

Returns a recent transaction list matching the given condition.

11.18.1 REQUEST

1. `String` - Hex string of search results. Default is 20
2. `String` - Hex string of skipped search results. Default is 0

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getRecentTransactions","params":[
↪"0x1","0x0"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.18.2 RESPONSE

`result` returns `Array` - The list of transaction object, see : see *app3.apis.getTransaction*:


```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getRecentTransactions",
  "result": [
    {
      "status": "0x01",
      "transactionHash":
↪"0xc32eed1b70c5a17172214be87e5771fdc1c2ce0396e23b0c96f9323efffaa791",
      "blockHash": "0x04bad3183936ef3f0be6fad6f1b26ea53cb9bb1f0d31e935dd23cddc2ab5d299
↪",
      "blockNumber": 1325386,
      "timestamp": "1556664769",
      "from": "0xf7afadb79a8b4579e3e113298eec866f7928c435",
      "to": "0xe6e11a5b9f794e398e7a832a764cb7728cfad359",
      "value": "1000000000000000000",
      "valueAPIS": "1.0",
      "gas": 220000,
      "gasPrice": "60000000000",
      "gasPriceAPIS": "0.00000006",
      "gasUsed": 213000,
      "fee": "12780000000000000",
      "feeAPIS": "0.01278",
      "mineralUsed": "12780000000000000",
      "mineralUsedMNR": "0.01278",
      "feePaid": "0",
      "feePaidAPIS": "0"
    }
  ]
}
```

11.19 apis_getRecentBlocks

Returns a block list of recent confirmed.

11.19.1 REQUEST

1. `String` - Maximum hex string of search results. Default is 20
2. `String` - Hex string of skipped search results. Default is 1

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getRecentBlocks","params":["0x1",
↪"0x0"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.19.2 RESPONSE

`result` returns `Array` - The list of 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` - `Number`: Balance of miner (10 blocks ago).
- `txTrieHash` 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.
- `coinbase` - `String`: The address of the beneficiary to whom the mining rewards were given.
- `coinbaseMask` - `String`: The mask of the coinbase.
- `rewardPoint` - `String`: Integer of the RP for this block of miner.
- `cumulativeRewardPoint` - `String`: Integer of the cumulative RP of the chain until this block.
- `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.
- `mineralUsed` - `Number`: The total used mineral 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.
- `logsBloom` 256 Bytes - `String`: The bloom filter for the logs of the block. `null` when its pending block.
- `mnHash` 32 Bytes - `String`: Hash of the masternodes
- `mnReward` - `Number`: Base amount of Masternode rewards
- `mnGenerals` - `Array`: Array of general masternodes.
- `mnMajors` - `Array`: Array of major masternodes.
- `mnPrivates` - `Array`: Array of private masternodes.
- `size` - `Number`: Integer the size of this block in bytes.


```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getRecentBlocks",
  "result": [
    {
      "number": 2018136,
      "hash": "0xd4e19ade32e4391c32dfbc29ee8c87bc191d72ca904179f9402651790e00d3e3",
      "parentHash":
↪"0x8ca6c4268fb54ef8acf53b944ea37c6c9f0af7214d2fd3d57c9312fde23a309d",
      "coinbase": "0x16b929423a857eebc802bb2b01a4a210998cfc29",
      "stateRoot": "0x16a70f060ba3d2f43c42146395d63b22c99c557dbd174cb2cb50d46f068ea002
↪",
      "txTrieHash":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
```


(continues on next page)

(continued from previous page)

```
      "receiptsTrieHash":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
      "rewardPoint": "191079955141446231419430344295608268161444500000000",
      "cumulativeRewardPoint":
↪"161072182513168577521270042219241988634704423646411097749 9",
      "gasLimit": 100000000,
      "gasUsed": 0,
      "mineralUsed": "0",
      "timestamp": "1562206769",
      "extraData": "0x4150495320706f7765726564205365727665 72",
      "rpSeed": "0x3aa9ba27dcbc9bb432e01bc09bf37f31179d4f89236b8c7883b9d7c13bd133b7",
      "nonce": "0x05f4f034afcf691f666d00",
      "txSize": 0,
      "transactions": [

      ],
      "size": 622
    }
  ]
}
```

11.20 apis_getTransactionByBlockNumberAndIndex

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

11.20.1 REQUEST

1. `String` - Hex string of block number.
2. `String` - Hex string of the transactions index position.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getTransactionByBlockNumberAndIndex
↪","params":["0x2710","0x1"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.20.2 RESPONSE

`result` returns `Object` - A transaction object, see *app3.apis.getTransaction*:

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getTransactionByBlockNumberAndIndex",
  "result": {
    "hash": "0x5949bca2a8e9819834e85e14253d189e7e94d142880f9ebef086d3220d071d1e",
    "nonce": "0x11d2",
    "blockHash": "0x93c601f19b570fb689eceb6b1551b8cb7e8b6d1f6cfe980fd139496cedcd7a7e",
    "blockNumber": "10000",
```

(continues on next page)

(continued from previous page)


```
    "timestamp": "1546141681",
    "from": "0xc5f590c1035ae780906514ff8e76dd86b89b97dc",
    "to": "0x866962b19d403a712f2c6bca390f9f295ba2dfe9",
    "value": "200000000000000000000000",
    "valueAPIS": "200,000",
    "gas": "4000000",
    "gasPrice": "50000000000",
    "gasPriceAPIS": "0.00000005",
    "feePaidAPIS": "0.2",
    "data":
↪"0x2c32ac980000000000000000000000000c5f590c1035ae780906514ff8e76dd86b89b97dc0000000000000000000000000
↪",
    "r": "0xc83ef4698b1f8b91b3f6fcb0675cddd10cf292a2966a0f062008eac81ee653d9",
    "s": "0x1a4035422b21cde64902b993f7aec9bd23ec6180ebd34241e35f518c0dab6939",
    "v": "0x1b"
  }
}
```


11.21 apis_getTransactionByBlockHashAndIndex

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

11.21.1 REQUEST

1. `String` - Hex string of block hash.
2. `String` - Hex string of the transactions index position.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getTransactionByBlockHashAndIndex",
↪"params":["0x93c601f19b570fb689eceb6b1551b8cb7e8b6d1f6cfe980fd139496cedcd7a7e","0x1
↪"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.21.2 RESPONSE

`result` returns `Object` - A transaction object, see *app3.apis.getTransaction*:


```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getTransactionByBlockHashAndIndex",
  "result": {
    "hash": "0x5949bca2a8e9819834e85e14253d189e7e94d142880f9ebef086d3220d071d1e",
    "nonce": "0x11d2",
    "blockHash": "0x93c601f19b570fb689eceb6b1551b8cb7e8b6d1f6cfe980fd139496cedcd7a7e",
    "blockNumber": "10000",
    "timestamp": "1546141681",
    "from": "0xc5f590c1035ae780906514ff8e76dd86b89b97dc",
```


(continues on next page)

(continued from previous page)

```
    "to": "0x866962b19d403a712f2c6bca390f9f295ba2dfe9",
    "value": "2000000000000000000000000",
    "valueAPIS": "200,000",
    "gas": "4000000",
    "gasPrice": "50000000000",
    "gasPriceAPIS": "0.00000005",
    "feePaidAPIS": "0.2",
    "data":
↪"0x2c32ac980000000000000000000000000c5f590c1035ae780906514ff8e76dd86b89b97dc000000000000000000000000
↪",
    "r": "0xc83ef4698b1f8b91b3f6fcb0675cddd10cf292a2966a0f062008eac81ee653d9",
    "s": "0x1a4035422b21cde64902b993f7aec9bd23ec6180ebd34241e35f518c0dab6939",
    "v": "0x1b"
  }
}
```

11.22 apis_getTransactionReceipt

Returns the receipt of a transaction by transaction hash.

11.22.1 REQUEST

1. `String` - Hex string of block hash.
2. `String` - Hex string of the transactions index position.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getTransactionReceipt","params":[
↪"3235deae37b6aba20a56d1958394314803a84fab4d861f09677767bc2329a897"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.22.2 RESPONSE

`result` 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.
- `timestamp`- `Number`: The unix timestamp for when the block was collated.
- `from` - `String`: Address of the sender.
- `to` - `String`: Address of the receiver. `null` when its a contract creation transaction.
- `toMask` - `String`: Mask of the receiver. `null` when its not registered.

- `contractAddress` - `String`: The contract address created, if the transaction was a contract creation, otherwise `null`.
- `gas` - `Number`: Gas provided by the sender.
- `gasPrice` - `String`: Gas price provided by the sender in atto.
- `gasPriceAPIS` - `String`: Gas price in APIS.
- `gasUsed`- `Number`: The amount of gas used by this specific transaction alone.
- `mineralUsed` - `String`: The amount of mineral used by this specific transaction alone.
- `mineralUsedMNR` - `String`: Used mineral in MNR.
- `feePaid` - `String`: Finally paid fee amount in atto.
- `feePaidAPIS` - `String`: Paid fee amount in APIS.
- `cumulativeGasUsed` - `Number`: The total amount of gas used when this transaction was executed in the block.
- `cumulativeMineralUsed` - `Number`: The total amount of mineral used when this transaction was executed in the block.
- `cumulativeMineralUsedMNR` - `Number`: Cumulative mineral in MNR
- `logs` - `Array`: Array of log objects, which this transaction generated.
- `internalTransaction` - `Array`: Array of internal transaction objects.


```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getTransactionReceipt",
  "result": {
    "status": "0x01",
    "transactionHash":
↪"0x3235deae37b6aba20a56d1958394314803a84fab4d861f09677767bc2329a897",
    "transactionIndex": 0,
    "blockHash": "0x93c601f19b570fb689eceb6b1551b8cb7e8b6d1f6cfe980fd139496cedcd7a7e",
    "blockNumber": 10000,
    "timestamp": "1546141681",
    "from": "0xc5f590c1035ae780906514ff8e76dd86b89b97dc",
    "to": "0x866962b19d403a712f2c6bca390f9f295ba2dfe9",
    "toMask": "platform@!",
    "value": "200000000000000000000000",
    "valueAPIS": "200,000",
    "nonce": 4561,
    "gas": 4000000,
    "gasPrice": "50000000000",
    "gasPriceAPIS": "0.00000005",
    "gasUsed": 940656,
    "fee": "47032800000000000",
    "feeAPIS": "0.0470328",
    "mineralUsed": "10000000000000",
    "mineralUsedMNR": "0.00001",
    "feePaid": "47022800000000000",
    "feePaidAPIS": "0.0470228",
    "cumulativeGasUsed": 940656,
    "cumulativeMineralUsed": "10000000000000",
    "cumulativeMineralUsedMNR": "0.00001",
```


(continues on next page)

(continued from previous page)

```
    "logs": [
      {
        "address": "0x866962b19d403a712f2c6bca390f9f295ba2dfe9",
        "topics": [
          "0xb78b065b331993702eab550134725ee93e37c65976734cd4317b7df694986489"
        ],
        "data":
↪"0x0000000000000000000000000c5f590c1035ae780906514ff8e76dd86b89b97dc0000000000000000000000000000000000000000000000005d40848
↪",
        "transactionHash":
↪"0x3235deae37b6aba20a56d1958394314803a84fab4d861f09677767bc2329a897",
        "logIndex": "0x",
        "blockNumber": 0,
        "transactionIndex": 0
      }
    ],
    "logsBloom":
↪"0x0000000000002000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "data":
↪"0x2c32ac980000000000000000000000000c5f590c1035ae780906514ff8e76dd86b89b97dc0000000000000000000000000000000000000000000000000
↪"
  }
}
```

11.23 apis_getTransactionCount

Get the numbers of transactions sent from this address.

11.23.1 REQUEST

`String` - The address to get the numbers of transactions from.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getTransactionCount","params":[
↪"0xc5f590c1035ae780906514ff8e76dd86b89b97dc"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.23.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getTransactionCount",
  "result": "0x0086d4"
}
```

11.24 apis_getMasternodeCount

Get the number of masternodes.

Masternode State :

- Earlybird - Masternode that can be joined through apis.mn. The nodes are joined to the day(10,800 blocks) before the round begins.
- Normal - Masternode joined through APIS Core within the first day(10,800 blocks) of the round
- Late - Masternode joined through the APIS Core after the normal participation period of the round

11.24.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getMasternodeCount","params":[]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.24.2 RESPONSE

`result` returns `Object` - The number of masternodes.

- `generalEarly` - `Number`: Number of General Earlybird Masternodes
- `majorEarly` - `Number`: Number of Major Earlybird Masternodes
- `privateEarly` - `Number`: Number of Private Earlybird Masternodes
- `generalNormal` - `Number`: Number of General Normal Masternodes
- `majorNormal` - `Number`: Number of Major Normal Masternodes
- `privateNormal` - `Number`: Number of Private Normal Masternodes
- `generalLate` - `Number`: Number of General Late Masternodes
- `majorLate` - `Number`: Number of Major Late Masternodes
- `privateLate` - `Number`: Number of Private Late Masternodes

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getMasternodeCount",
  "result": {
    "generalEarly": 3390,
    "majorEarly": 2907,
    "privateEarly": 1967,
    "generalNormal": 0,
    "majorNormal": 0,
    "privateNormal": 2,
    "generalLate": 1,
    "majorLate": 0,
    "privateLate": 2
  }
}
```

11.25 apis_sendRawTransaction

Sends an already signed transaction.

11.25.1 REQUEST

1. `String` - Signed transaction data in HEX format

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":4,"method":"apis_sendRawTransaction","params":[
↪"0xf87380850ba43b740083030d40943535353535353535353535353535353535353535353535808a152d02c7e14af6800000802
↪"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.25.2 RESPONSE

`result` returns `String` - The transaction hash

```
{
  "id": 4,
  "jsonrpc": "2.0",
  "method": "apis_sendRawTransaction",
  "result": "0xac05c474f7ff71cd8d91169d41e0efb92a5cbc7f9ec1bdb3ba8ec1c3d6c14cd5"
}
```

11.26 apis_sendTransaction

Sends a transaction to the network. The sender address must be unlocked.

11.26.1 REQUEST

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

- `from` - `String`: The address for the sending account.
- `to` - `String`: (optional) The destination address of the message, left undefined for a contract-creation transaction.
- `value` - `String`: (optional) The value transferred for the transaction in atto, also the endowment if it's a contract-creation transaction in hex format.
- `gas` - `String`: (optional, default: To-Be-Determined) The amount of gas to use for the transaction (unused gas is refunded) in hex format.
- `gasPrice` - `String`: (optional) The price of gas in hex format for this transaction in atto , defaults to *app3.apis.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` - `String`: (optional) Hex string of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":3,"method":"apis_sendTransaction","params":[{"from":
↪"0x71f81feeb2bc39456455e4793c3ed9286132f56b","gas":"0x30d40","to":
↪"0x3535353535353535353535353535353535353535","value":"0x0","gasPrice":
↪"0x0000000ba43b7400"}]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.26.2 RESPONSE

`result` returns `String` - The transaction hash

```
{
  "id": 3,
  "jsonrpc": "2.0",
  "method": "apis_sendTransaction",
  "error": {
    "code": -1,
    "message": "Not enough cash: Require: 10000000000000000, Sender cash: 0"
  }
}

{
  "id": 3,
  "jsonrpc": "2.0",
  "method": "apis_sendTransaction",
  "error": {
    "code": -1,
    "message": "Sender account is locked"
  }
}

{
  "id": 3,
  "jsonrpc": "2.0",
  "method": "apis_sendTransaction",
  "result": "0xac05c474f7ff71cd8d91169d41e0efb92a5cbc7f9ec1bdb3ba8ec1c3d6c14cd5"
}
```

11.27 apis_sign

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

11.27.1 REQUEST

1. `String` - Data to sign in Hex format. The String can be converted using app3.utils.utf8ToHex.

2. `String` - Address to sign data with.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":2,"method":"apis_sign","params":[
↪"0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01","0x48656c6c6f20776f726c64"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.27.2 RESPONSE

`result` returns `String` - The signature.

```
{
  "id": 2,
  "jsonrpc": "2.0",
  "method": "apis_sign",
  "result":
↪"0xf4870babd44ad7b26f99ff1d8249d9aeb8aabe7585f81cd8bed15c04c1aafd28163e6a418ccc92544c4257da634f787a
↪"
}
```

11.28 apis_call

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

11.28.1 REQUEST

1. `Object` - A transaction object see *app3.apis.sendTransaction*, with the difference that for calls the `from` property is optional as well.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_call","params":[{"to":
↪"0x866962b19d403a712f2c6bca390f9f295ba2dfe9","data":
↪"0x2fc1f190000000000000000000000000000000000000000000000000000000000000000000000000000000"},
↪"latest"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.28.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_call",
  "result": "0x00000000000000000000000c5f590c1035ae780906514ff8e76dd86b89b97dc"
}
```

11.29 apis_estimateGas

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

11.29.1 REQUEST

1. `Object` - A transaction object see *app3.apis.sendTransaction*, with the difference that for calls the `from` property is optional as well.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":2,"method":"apis_estimateGas","params":[{"from":
↪"0x71f81feeb2bc39456455e4793c3ed9286132f56b","gas":"0x30d40","to":
↪"0x3535353535353535353535353535353535353535","value":"0x0","keystorepassword":"pA
↪$sw0rD"}]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.29.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_estimateGas",
  "result": "0x0000000000030d40"
}
```

11.30 apis_getLogs

Gets past logs, matching the given options.

11.30.1 REQUEST

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

- `fromBlock` - `String`: Hex string of the earliest block
- `toBlock` - `String`: Hex string of the latest block
- `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']]`

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"apis_getLogs","params":[{"fromBlock":
↪"0x1e8480","address":"0x866962b19d403a712f2c6bca390f9f295ba2dfe9","topics":[]}]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.30.2 RESPONSE

result' 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.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "apis_getLogs",
  "result": [
    {
      "status": "0x01",
      "transactionHash":
↪"0x5168279997a492bb89dac17409f74709f3def7439c8460d69bf0ed28ebeabc81",
      "transactionIndex": 0,
      "blockHash": "0x64e4f0b3191012e40f4e89f1e1b5beec98fa9480db1ae241288e4ef99b4f1e61
↪",
      "blockNumber": 2000077,
      "timestamp": "1562062297",
      "from": "0xc5f590c1035ae780906514ff8e76dd86b89b97dc",
      "to": "0x866962b19d403a712f2c6bca390f9f295ba2dfe9",
      "value": "0",
      "valueAPIS": "0",
      "nonce": 34499,
      "gas": 4000000,
      "gasPrice": "50000000000",
      "gasPriceAPIS": "0.00000005",
      "gasUsed": 347670,
      "fee": "17383500000000000",
      "feeAPIS": "0.0173835",
      "mineralUsed": "1600000000000000",
      "mineralUsedMNR": "0.0016",
      "feePaid": "15783500000000000",
      "feePaidAPIS": "0.0157835",
      "cumulativeGasUsed": 347670,
```

(continues on next page)

(continued from previous page)


```
      "cumulativeMineralUsed": "1600000000000000",
      "cumulativeMineralUsedMNR": "0.0016",
      "logs": [
        {
          "address": "0x866962b19d403a712f2c6bca390f9f295ba2dfe9",
          "topics": [
            "0x19b6dcee97bdeda88d78fa94793f2271c3d9eceba8f8c02a04bff5d632ca3742"
          ],
          "data":
↪"0x000000000000000000000000005527867bad124260b636c6f5991147e18ddf6ec1000000000000000000000000000f6d8cf6
↪",
          "transactionHash":
↪"0x5168279997a492bb89dac17409f74709f3def7439c8460d69bf0ed28ebeabc81",
          "logIndex": "0x",
          "blockNumber": 0,
          "transactionIndex": 0
        }
      ],
      "logsBloom":
↪"0x00000000000020000000010000000000000002000000000000000000000000000000000000000000000000000000000000000
↪",
      "data":
↪"0x0c05f1890000000000000000000000000000f6d8cf6194972fe31d047631a12977f899e4cfa000000000000000000000000000
↪"
    }
  ]
}
```


11.31 apis_call

Create a new account on the node that App3js is connected to with its provider. The RPC method used is `personal_newAccount`. It differs from app3.apis.accounts.create() where the key pair is created only on client and it's up to the developer to manage it.

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

11.31.1 REQUEST

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

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"personal_newAccount","params":["pAs$w0Rd
↪"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.31.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "personal_newAccount",
  "result": "0xd174cf67355c1df76c18049d5f08637c32836387"
}
```

11.32 personal_sign

Signs data using a specific account.

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

11.32.1 REQUEST

1. `String` - Data to sign in Hex format. The String can be converted using app3.utils.utf8ToHex.
2. `String` - Address to sign data with.
3. `String` - The password of the account to sign data with.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"personal_sign","params":[
↪"0x48656c6c6f20776f726c64","0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01","pAs$w0Rd"]}
↪' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.32.2 RESPONSE

`result` returns `String` - The signature.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "personal_sign",
  "result":
↪"0xf4870babd44ad7b26f99ff1d8249d9aeb8aabe7585f81cd8bed15c04c1aafd28163e6a418ccc92544c4257da634f787a
↪"
}
```

11.33 personal_ecRecover

Recovers the account that signed the data.

11.33.1 REQUEST

1. `String` - Data that was signed. If String it will be converted using app3.utils.utf8ToHex.
2. `String` - The signature in Hex format.
3. `String` - The password of the account to sign data with.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"personal_ecRecover","params":[
↪"0x48656c6c6f20776f726c64",
↪"f4870babd44ad7b26f99ff1d8249d9aeb8aabe7585f81cd8bed15c04c1aafd28163e6a418ccc92544c4257da634f787aeb
↪"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.33.2 RESPONSE

`result` returns `String` - The account.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "personal_ecRecover",
  "result": "0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01"
}
```

11.34 personal_signTransaction

Signs a transaction. This account needs to be unlocked.

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

11.34.1 REQUEST

1. `Object` - The transaction data to sign *app3.apis.sendTransaction()* for more.
2. `String` - The password of the `from` account, to sign the transaction with.

```
curl -X POST \
    -H "Content-Type: application/json" \
```

(continues on next page)

(continued from previous page)

```
    --data '{"jsonrpc":"2.0","id":1,"method":"personal_signTransaction","params":[{
↪"from":"0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01","gasPrice":"0xba43b7400","gas":
↪"0x30d40","to":"0x3535353535353535353535353535353535353535","value":
↪"0xde0b6b3a7640000","data":""},"pAs$w0Rd"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.34.2 RESPONSE

`result` returns `Object` - The RLP encoded transaction. The `raw` property can be used to send the transaction using *app3.apis.sendSignedTransaction*.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "personal_signTransaction",
  "result": {
    "raw":
↪"0xf87180850ba43b740083030d40943535353535353535353535353535353535353535353580880de0b6b3a76400008025a00
↪",
    "tx": {
      "hash": "0x185da901f30171b573d06ffdf1cbe8aa7e624eb3d207e4c6cede5c28c0fd948c",
      "nonce": "0x",
      "from": "0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01",
      "to": "0x3535353535353535353535353535353535353535",
      "value": "1000000000000000000",
      "valueAPIS": "1",
      "gas": "200000",
      "gasPrice": "50000000000",
      "gasPriceAPIS": "0.00000005",
      "feePaidAPIS": "0.01",
      "data": "",
      "r": "0x0975caffb4cb5c6f9dfe97315d3c448926ee0d3be85a44c1006be525d38ef5a3",
      "s": "0x1b8c0d9ac65615823dbdc0dfd398f5807d0582f490b2b408224e4803d46c3b69",
      "v": "0x1b"
    }
  }
}
```

11.35 personal_unlockAccount

Unlocks the given account.

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

11.35.1 REQUEST

1. `address` - `String` - The account address.
2. `password` - `String` - The password of the account.

3. `unlockDuration` - `Number` - The duration seconds for the account to remain unlocked.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"personal_unlockAccount","params":[
↪"0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01","pAs$w0Rd",600]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.35.2 RESPONSE

`result` returns `Object` - The RLP encoded transaction. The `raw` property can be used to send the transaction using *app3.apis.sendSignedTransaction*.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "personal_unlockAccount",
  "result": true
}
```

11.36 personal_lockAccount

Locks the given account.

11.36.1 REQUEST

1. `address` - `String` - The account address.

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"personal_lockAccount","params":[
↪"0xb8ce9ab6943e0eced004cde8e3bbed6568b2fa01"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.36.2 RESPONSE

`result` returns `boolean` - True if the account got locked successful otherwise false.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "personal_lockAccount",
  "result": true
}
```

11.37 personal_listAccounts

Returns a list of accounts the node controls by using the provider and calling the RPC method `personal_listAccounts`.

The results are the same as RPC method `apis_accounts`.

11.37.1 REQUEST

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"personal_listAccounts","params":[]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.37.2 RESPONSE

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

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "personal_listAccounts",
  "result": [
    {
      "address": "aaad4d1b7bcb7ac91f33ab75027257562ac06b5e",
      "index": "0",
      "aAPIS": "28008000000000",
      "aMNR": "0",
      "nonce": "0",
      "APIS": "0.000028008",
      "MNR": "0"
    },
    {
      "address": "de9e9e4c0b94469ce9913cb040ec3d8f38bbdb64",
      "index": "1",
      "aAPIS": "500000000000000000000",
      "aMNR": "20000000000000000",
      "nonce": "0",
      "APIS": "5,000",
      "MNR": "0.02"
    }
  ]
}
```

11.38 personal_importRawKey

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.

11.38.1 REQUEST

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

```
curl -X POST \
    -H "Content-Type: application/json" \
    --data '{"jsonrpc":"2.0","id":1,"method":"personal_importRawKey","params":[
↪"0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe870c","aaaa"]}' \
    --user RPC-ID:RPC-Password http://127.0.0.1:45678
```

11.38.2 RESPONSE

`result` returns `String` - The address of the account.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "method": "personal_importRawKey",
  "result": "9d158a9dc16c2c03aec4e6cbc8d29d7734a8b455"
}
```

Index

C

`contract deploy`, 54

J

`JSON interface`, 51

N

`npm`, 3