Read the Docs Template Documentation

Release 1.0

Read the Docs

Feb 07, 2020

Contents

Table of Contents:

CHAPTER 1

Example

`czr.js` is a library for interacting with canonchain nodes.

1.1 Install

First, you need to put `czr.js` into your project.

You can do this using:

```
npm install czr
```

After that, you need to start the local Canonchain node and instantiate `czr`.

Canonchain download: http://dev.canonchain.com/download.html

Canonchain command line: https://canonchain.readthedocs.io/zh/latest/source/Command-Line-Interfaces.html

Port used by default `127.0.0.1:8765`

1.2 Use

```
let Czr = require("czr");
let czr = new Czr();// Use by default 127.0.0.1:8765
```

(continues on next page)

(continued from previous page)

```
// If you want to modify the port, please set the following
// Example
// const options = {
//     host: "127.0.0.1",
//     port: 8888
// };
// let czr = new Czr(options);

//Now you can use the czr object
//Example
czr.request.status().then(function (res) {
    console.log(`status`,res);
}).catch(function(error){
    console.log("accountList catch",error)
});
```

CHAPTER 2

Accounts

2.1 create

Create a new account.

```
czr.accounts.create(password)
    .then(res => {})
    .catch()
```

2.1.1 Parameters

2.1.2 Results

keystore file of a new account.

2.1.3 Example

```
czr.accounts.create(123456).then(res => {
    console.log("Create account result\n", res);//res.account
    console.log(`res.account:${JSON.stringify(res)}`);
}).catch(err => {
```

(continues on next page)

(continued from previous page)

```
    console.log("err===>", err);
});
```

2.2 validateAccount

Validate if the `keystore` file is decodable by the password.

```
czr.accounts.validateAccount(keystore,password)
    .then(res => {})
    .catch()
```

2.2.1 Parameters

2.2.2 Results

If the password can decode the `keystore` file, return `true`. Otherwise, return `false`.

2.2.3 Example

```
czr.accounts.validateAccount(keystore,123456).then(res => {
    console.log("Result\n", res);
}).catch(err => {
    console.log("err===>", err);
});
```

2.3 decrypt

Derive the privateKey from `keystore` file and account password.

```
czr.accounts.decrypt(keystore,password)
    .then(res => {})
    .catch()
```

2.3.1 Parameters

2.3.2 Results

If the password can decode `keystore` file, return the privateKey. Otherwse, report error.

2.3.3 Example

```
czr.accounts.decrypt(keystore,123456).then(res => {
    console.log("Result\n", res);
}).catch(err => {
    console.log("err===>", err);
});
```

2.4 sign

Sign the transaction using privateKey.

```
czr.accounts.sign(transHash,privateKey)
    .then(res => {})
    .catch()
```

2.4.1 Parameters

2.4.2 Returns

Signed transaction

2.4.3 Example

```
czr.accounts.decrypt(account_keystore, '1234qwer')
    .then(privateKey => {
        console.log('privateKey', privateKey)
        let blockHash='5E844EE4D2E26920F8B0C4B7846929057CFCE48BF40BA269B173648999630053';
        czr.accounts.sign(blockHash, privateKey)
            .then(signature => {
                console.log('signature', signature)
            })
```

(continues on next page)

(continued from previous page)

```
        .catch(console.error)
})
.catch(console.error)
```

CHAPTER 3

RPC requests

3.1 accountBalance

Get the balance of an account.

```
request.accountBalance
```

3.1.1 Parameters

3.1.2 Returns

Error code

3.2 accountBlockList

Get the list of blocks from an account.

Note：set `rpc_control` to be `true` when the node starts

```
request.accountBlockList
```

3.2.1 Parameters

3.2.2 Returns

Error code

3.3 accountChangePwd

Change the account password.

Note: set `rpc_control` to be `true` when the node starts

```
request.accountChangePwd
```

3.3.1 Parameters

3.3.2 Returns

Error code

3.4 accountCode

Get code from an account if it is a contract account.

```
request.accountCode
```

3.4.1 Parameters

3.4.2 Returns

Error code

3.5 accountCreate

Create an account.

Note: set `rpc_control` to be `true` when the node starts

```
request.accountCreate
```

3.5.1 Parameters

3.5.2 Returns

Error code

3.6 accountExport

Export an account

```
request.accountExport
```

3.6.1 Parameters

3.6.2 Returns

Error code

3.7 accountImport

Import an account

Note: set `rpc_control` to be `true` when the node starts

```
request.accountImport
```

3.7.1 Parameters

3.7.2 Returns

Error code

3.8 accountList

```
request.accountList
```

3.8.1 Returns

3.9 accountLock

Lock an account

Note: set `rpc_control` to be `true` when the node starts

```
request.accountLock
```

3.9.1 Parameters

3.9.2 Returns

Error code

3.10 accountRemove

Remove an account.

Note: set `rpc_control` to be `true` when the node starts

```
request.accountRemove
```

3.10.1 Parameters

3.10.2 Returns

Error code

3.11 accountUnlock

Unlock an account

Note: set `rpc_control` to be `true` when the node starts.

```
request.accountUnlock
```

3.11.1 Parameters

3.11.2 Returns

Error code

3.12 accountValidate

Validate account format

```
request.accountValidate
```

3.12.1 Parameters

3.12.2 Returns

3.13 accountsBalances

Check balance for multiple accounts

```
request.accountsBalances
```

3.13.1 Parameters

3.13.2 Returns

Error code

3.14 call

Get contract states

```
request.call
```

3.14.1 Parameters

3.14.2 Returns

Error code

3.15 estimateGas

estimate gas consumption for a transaction

```
request.estimateGas
```

3.15.1 Parameters

3.15.2 Returns

Error code

3.16 generateOfflineBlock

Generate an unsigned block.

Note: set `rpc_control` to be `true` when the node starts.

```
request.generateOfflineBlock
```

3.16.1 Parameters

3.16.2 Returns

Error code

3.17 getBlock

Get content of a block.

```
request.getBlock
```

3.17.1 Parameters

3.17.2 Returns

Error code

3.18 getBlockState

Get state of a block.

```
request.getBlockState
```

3.18.1 Parameters

3.18.2 Returns

Error code

3.19 getBlockStates

Get block states in batch.

```
request.getBlockStates
```

3.19.1 Parameters

3.19.2 Returns

Error code

3.20 logs

The event logs of smart contract execution.

```
request.logs(obj)
```

3.20.1 Parameters

3.20.2 Returns

```
// Success
{
    "code": 0,
    "msg": "OK",
    "logs": [
        {
            "address": "czr_3DG8FjYSAqkBNubcSVAAjAtSQ9Q2tWVNwPS8VHQ55XwWG4DsTS",
            "data": "0000000000000000000000000000000000000000000000000000000000000000",
            "topics": [
                "260823607ceaa047acab9fe3a73ef2c00e2c41cb01186adc4252406a47d73446"
            ]
        },
        {
            "address": "czr_3DG8FjYSAqkBNubcSVAAjAtSQ9Q2tWVNwPS8VHQ55XwWG4DsTS",
            "data": "0000000000000000000000000000000000000000000000000000000000000001",
            "topics": [
                "260823607ceaa047acab9fe3a73ef2c00e2c41cb01186adc4252406a47d73446"
            ]
        }
    ]
}

// Failed
{
    "code": 3,
    "msg": "Invalid account"
}
```

3.21 sendBlock

Send a block.

Note: set `rpc_control` to be `true` when the node starts.

```
request.sendBlock
```

3.21.1 Parameters

3.21.2 Returns

Error code

3.22 sendOfflineBlock

Send a signed block. The parameters are derived from *generateOfflineBlock* return fields. Returns the block hash.

Note: set `rpc_control` to be `true` when the node starts.

```
request.sendOfflineBlock
```

3.22.1 Parameters

3.22.2 Returns

Error code

3.23 signMsg

Sign a message

```
request.signMsg
```

3.23.1 Parameters

3.23.2 Returns

Error code

3.24 stableBlocks

Retrieve the stabled blocks for a specific mci value.

```
request.stableBlocks
```

3.24.1 Parameters

3.24.2 Returns

Error code

3.25 status

Retrieve the current status of DAG on the node

```
request.status
```

3.25.1 Returns

3.26 version

Acquire the current node version, rpc interface version, and database version.

```
request.version
```

3.26.1 Returns

3.27 witnessList

Retrieve the list of witnesses.

```
request.witnessList
```

3.27.1 Returns

CHAPTER 4

Contract

4.1 Usage

`Contract` object facilitates the interaction of the developers to the smart contracts on Canonchain.

The developer has to provide json interface for a contract.

An example of JSON interface for a contract is as the following:

```
[
        {
                "constant": true,
                "inputs": [],
                "name": "xxx",
                "outputs": [
                        {
                                "name": "",
                                "type": "uint256"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        },
```

(continues on next page)

(continued from previous page)

```
        ...
]
```

4.1.1 Rules

- The prameters in `[]` is optional. Other pamameters are mandatory.
- The variable `myContract` in this document is a contract object.

4.2 Create a contract

Create a contract object (include all the methods defined in JSON interface).

```
let myContract = new czr.Contract(jsonInterface[, account][, options])
```

4.2.1 Prameters

- `jsonInterface` <Object> : JSON interface
- `account` <String> : Contract address
  - it can also be setup later by using `myContract.options.account = 'czr_account'`
- `options` <Object>: Optional parameters for `call` and `sendBlock` methods.
  - `from` <String>: Sender address.
  - `gas_price` <String> : Gas price, Unit: 1*10-18 CZR.
  - `gas` <String> | <Number> : Gas uplimit for the block.
  - `data` <String> : Contract data.

4.2.2 Returns

`Object`: Contract object and all its methods.

4.2.3 Example

```
let myContract = new czr.Contract(
    [...],
    'czr_contract_address',
    {
        from: 'czr_account',
        gas_price: '20000000000',
        gas:'2000000',
        data:''
    }
);
```

4.3 myContract properties

- options
  - options.account
  - options.jsonInterface
- methods

4.4 options

```
myContract.options;
```

4.4.1 Properties

- account <String>: Contract account.
- jsonInterface <Array>: JSON interface.
- data <String>: Contract data. Used for contract deployment.
- from <String>: Sender account. Null if there is no sender.
- gas_price <String>: Gas price.
- gas <String> | <Number>: Gas limit of the block.

4.4.2 Example

```
// get
myContract.options;
> {
    account: 'contract_account',
    jsonInterface: [...],
    from: 'czr_account',
    gas_price: '10000000000000',
    gas: 1000000
}

// set
myContract.options.account = 'contract_account';
myContract.options.from = 'czr_account';
myContract.options.gas_price = '2000000000';
myContract.options.gas = 5000000;
```

4.5 options.account

The account to interact with. The contract object is to use this account as the `'to'` value for all transactions.

4.5.1 Properties

`myContract.options.account` <String> | <Null>: The contract account. `null` if not been set yet.

4.5.2 Example

```
//get
myContract.options.account;
> 'czr_contract_address'

// set
myContract.options.account = 'czr_contract_address';
```

4.6 options.jsonInterface

Contract JSON interface.

4.6.1 Properties

`myContract.options.jsonInterface` <Array>: The contract JSON interface

Contract instance methods will be regenerated if this value is reset;

4.6.2 Example

```
myContract.options.jsonInterface;
> [
    {
        "type":"function",
        "name":"foo",
        "inputs": [{"name":"a","type":"uint256"}],
        "outputs": [{"name":"b","type":"address"}]
    }
]

// set
myContract.options.jsonInterface = [...];
```

4.7 myContract.methods properties

`myContract.methods` returns all methods of the contract, where every method will create an object to `call`, `sendBlock` and `encodeABI`.

4.8 myContract methods

- clone
- deploy
- methods
  - methods.myMethod.call

- methods.myMethod.sendBlock
- methods.myMethod.encodeABI

4.9 clone

Clone a contract instance. The cloned instance is completely independent to the original one.

```
myContract.clone(); // No parameters
```

4.9.1 Returns

The new contract instance.

4.9.2 Example

```
let contract1 = new czr.Contract(abi, account, {gas_price: '12345678', from: fromAddress}
↪);

let contract2 = contract1.clone();
contract2.options.account = account2;

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

4.10 deploy

The contract is deployed by calling this method.

```
myContract.deploy({
  data: '',
  arguments: []
});
```

4.10.1 Parameters

- `options` <Object>: Deployment options.
  - `data` <String>: Contract code.
  - `arguments` <Array> (Optional) : Contract constructor paramenters in the deployment.

4.10.2 Returns

Object:

- `arguments` <Array>: The parameters passed to the contractor constructor. Their value may have been changed.
- `sendBlock` <Function>: Deploy the contract to Canonchain.
- `encodeABI` <Function>: Enocde the contract ABI, i.e. data and arguments.

4.10.3 Example

sendBlock

```
// sendBlock promise
myContract
  .deploy({
    data: bytecode
  })
  .sendBlock({
    from: 'czr_account',
    gas: 3000000,
    gas_price: '1000000000'
  })
  .then(data => {
    console.log('data', data);
  })
  .catch(function(error) {
    console.log('error', error);
  });

// sendBlock callback
myContract
  .deploy({
    data: bytecode
```

(continues on next page)

(continued from previous page)

```
  })
  .sendBlock(
    {
      from: 'czr_account',
      gas: 3000000,
      gas_price: '1000000000'
    },
    function(error, transactionHash) {
      console.log('error ==> ', error);
      console.log('transactionHash ==> ', transactionHash);
    }
  );
```

encodeABI

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

> '0x12345...0000012345678765432'
```

4.11 methods

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

`myMethod` is a method name in JSON interface, and it can be called in the following ways:

- Name: `myContract.methods.myMethod(123)`
- Name with parameters: `myContract.methods['myMethod(uint256)'](123)`
- Signature: `myContract.methods['0x58cf5f10'](123)`

it is allowed to have methods with same name but different types of parameters.

4.11.1 Parameters

The parameters are from the definition of JSON interface.

4.11.2 Returns

Object:

- `arguments` <Array>: The parameters passed to the contractor constructor. Their value may have been changed.
- `call` <Function>: Call a get method of the contract without sending a blcok.
- `sendBlock` <Function>: Send a block to Canonchain.
- `encodeABI` <Function>: Enocde the method ABI.

4.12 methods.myMethod.call

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

4.12.1 Parameters

- options <Object>: Options for call.
  - from <String>: Sender' s account
- callback <Function>: The first parameter of the callback is error object, and the second parameter of the callback is contract execution result if there is no error.

4.12.2 Returns

`Promise` object:

```
//promise
myContract.methods.testCall1().call()
    .then(data => {
        console.log('testCall1 data', data)
    })
    .catch(function (error) {
        console.log('error', error)
    });

//callback
myContract.methods.testCall1().call(function(error, result){
    ...
})
```

4.13 methods.myMethod.sendBlock

Send a block to the contract and execute the transaction in block.

4.13.1 Parameters

- `options` <Object>: Options for sendBlock.
  - `from` <String>: The sender account.
  - `gas_price` <String>: Gas price for the transaction.
  - `gas` <String>| <Number>: Gas limit.
  - `value Number | String | BN | BigNumber`: Value in the transaction.
- `callback` <Function>: The first parameter of the callback is error object, and the second parameter of the callback is contract execution result if there is no error.

4.13.2 Returns

<Promise> Return the transaction receipt.

4.13.3 Example

```
// promise
myContract.methods.myMethod(123).sendBlock({from: 'czr_account'})
    .then(function(receipt){
        // transaction receipt
    });

// callback
myContract.methods.myMethod(123).sendBlock({from: 'czr_account'}, function(error,␣
↪transactionHash){
    ...
});
```

4.14 methods.myMethod.encodeABI

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

No parameters

4.14.1 Returns

<String>: ABI bytecode.

4.14.2 Example

```
let encodeABIData = myContract.methods.myMethod(123).encodeABI();
console.log(encodeABIData);
//'0x58cf5f10000000000000000000000000000000000000000000000000000000000000007B'
```

CHAPTER 5

Utils

5.1 toBigNumber

Convert a value to `bignumber`

```
czr.utils.toBigNumber(value)
```

5.1.1 Parameters

5.1.2 Results

A value in Bignumber type.

```
BigNumber { s: 1, e: 0, c: [ 2 ] }
```

5.1.3 Example

```
const bigVal = czr.utils.toBigNumber(2);
```

5.2 isBigNumber

Check if the value is a `bignumber` type.

```
czr.utils.isBigNumber(value)
```

5.2.1 Parameters

5.2.2 Results

If the value is in Bignumber type, return `true`. Otherwise, return `false`.

5.2.3 Example

```
const isBig = czr.utils.isBigNumber(2);
```

5.3 encode

Encode an object.

```
czr.utils.encode.parse(parm);
```

5.3.1 Parameters

5.3.2 Returns

Encoded bytecode.

5.3.3 Example

```
let parm = {
    functionName: "constructor",
    args: ["100000000000000000000000000000", "canonChain", "CZR"]
};
let data = czr.utils.encode.parse(parm);

console.log("******************** data **********************");
```

(continues on next page)

(continued from previous page)

```
console.log(data);

// Returns
//{ functionName: 'constructor',
  args: [ '10000000000000000000000000000000', 'canonChain', 'CZR' ],
  funABI:
   { inputs: [ [Object], [Object], [Object] ],
     payable: false,
     stateMutability: 'nonpayable',
     type: 'constructor' },
  data:
   '60806040...' }
```

5.4 decode

Decode a bytecode according to an ABI.

```
utils.decode.parse(bytecode, abi);
```

5.4.1 Parameters

5.4.2 Result

Decoded object.

5.4.3 Example

```
let abi = {
    constant: true,
    inputs: [],
    name: "name",
    outputs: [{ name: "", type: "string" }],
    payable: false,
    stateMutability: "view",
    type: "function",
}
```

(continues on next page)

(continued from previous page)

```
let name_response = "0x" +
↪"0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000
↪";

let nameInfo2 = czr.utils.decode.parse(name_response, abi);
//[ { name: '', type: 'string', value: 'canonChain' } ]
```

5.5 encodeAccount

Convert a hexdecimal value to a Canonchain account address in `czr_xxx` format.

```
czr.utils.encodeAccount(value);
```

5.5.1 Parameters

5.5.2 Returns

Canonchain account address in `czr_xxx` format.

5.5.3 Example

```
let czrAcc = czr.utils.encodeAccount(
↪"B5F327E3F07F2C94DADCDB6D122ADDAFD3AA3AC9507E8F8368F9AD3E6A378798")
console.log(czrAcc);//czr_4KsqkcZCs6i9VU2WUsiqTU8M6i3WYpVPFMcMXSkKmB92GJvYt1
```

5.6 decodeAccount

Convert a Canonchan account address in `czr_xxx` format to hexdecimal value.

```
czr.utils.decodeAccount(value);
```

5.6.1 Parameters

5.6.2 Results

Hexdecimal value

5.6.3 Example

```
try {
    let acc = czr.utils.decodeAccount('czr_
↪4KsqkcZCs6i9VU2WUsiqTU8M6i3WYpVPFMcMXSkKmB92GJvYt1');
    console.log(acc)
} catch (error) {
    console.log(error)
}
```

5.7 fromCan

Convert an amount of tokens in uint of `can` to an equivalent amount in unit of `czr`.

```
czr.utils.fromCan(czrVal)
```

5.7.1 Parameters

5.7.2 Result

Amount in the uint of `czr`.

5.7.3 Example

```
let val = czr.utils.fromCan(2);//0.000000000000000002
```

5.8 fromCanToken

Convert an amount of tokens in uint of `can` to an equivalent amount in specific precision.

```
czr.utils.fromCan(value,precision)
```

5.8.1 Parameters

5.8.2 Results

Amount in a specific precision.

5.8.3 Example

```
let val = czr.utils.fromCanToken(2,1000000000000000000);//0.000000000000000002
```

5.9 toCan

Convert an amount of tokens in uint of `czr` to an equivalent amount in unit of `can`.

```
czr.utils.toCan(value)
```

5.9.1 Parameters

5.9.2 Results

Amount in the uint of `can`.

5.9.3 Example

```
let val = czr.utils.toCan(2);//2000000000000000000
```

CHAPTER 6

Abi

6.1 encodeEventSignature

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

```
czr.abi.encodeEventSignature(eventName);
```

6.1.1 Parameters

6.1.2 Returns

String - The ABI signature of the event.

6.1.3 Example

```
let sign1 = czr.abi.encodeEventSignature('myEvent(uint256,bytes32)');

let opt2 = {
    name: 'myEvent',
    type: 'event',
    inputs: [{
```

(continues on next page)

(continued from previous page)

```
        type: 'uint256',
        name: 'myNumber'
    }, {
        type: 'bytes32',
        name: 'myBytes'
    }]
};
let sign2 = czr.abi.encodeEventSignature(opt2);
```

CHAPTER 7

Index and Search

- genindex
- search