burnweb

Comps ITL Pte. Ltd.

May 03, 2021

DOCUMENTATION

burnweb is a library for interacting with BURN blockchain via REST API.

Contents:

CHAPTER

ONE

GETTING STARTED

burnweb is a library for interacting with BURN blockchain via REST API.

1.1 Install burnweb

- Install the *burnweb* package by npm command

```
npm install git+https://github.com/{your_repository}/burnweb.git
```

- Create a burnweb instance with a provider (BURN endpooint) specified.

```
// For js
var BurnWeb = require('burnweb');

// For ts
import BurnWeb from 'burnweb';;

// Set BURN endpoint URL to initialize
var burnweb = new BurnWeb('http://localhost:8545');

// or, specify private key at the same time
var burnweb = new BurnWeb('http://localhost:8545', privateKey);
```

CHAPTER

TWO

TRANSACTION HASH AND SIGNATURE

We use the Ethereum compatible transaction data structure. Transaction hash will be calculated from the following items.

- nonce
- gasPrice
- gasLimit
- to
- value
- data
- chainId
- 0
- 0

Using RLP (Recursive Length Prefix) encoding, pack this structure into bytes. Hash the bytes value with `keccak256` function to get a hash value for signature. This hash is called Transaction Hash (or Transaction ID).

Sign the hash value using ECDSA (Elliptic Curve Digital Signature Algorithm) to derive a signature. This signature is represented as 3 values (`v`, `r`, `s`). The value `v` is calculated by the following formula. This will help avoiding "Replay Attach" by checking the value `v` is valid for the network or not.

```
v: chainId * 2 + 8 + v
```

Concatenate the value `v`, `r`, `s` to get the signature string.

2.1 nonce

`nonce` is a positive integer value unique for the same transaction data from the same address. It's necessary to be unique, but not necessary to be consecutive (Ethereum doesn't allow skipping numbers for noce, but BURN does). `burnweb` uses system time in ms as a nonce of sending transaction.

2.2 gasPrice, gasLimit

Ethereum compatible tools, such as MetaMask, specifies `gasPrice` and `gasLimit` parameters in the transaction data. To keep the compatibility with it, BURN includes these parameters in Transaction Hash deriving process. The transactions sent via `burnweb` or `BURN REST API` doesn't have *gasPrice* and `gasLimit` parameters, and BURN always regards them as `0`.

2.3 to, value, data

Ethereum compatible tools handle the parameters `to` and `value` differently for transfering ETH or other ERC20 tokens.

- **ETH (or a native token for the network):**
  - to: The address to which a sender transfer the token.
  - value: The amount of ETH to transfer.
  - data: empty or '0x'

Note: On Ethereum, a sender can specify `data` when sending ETH to a smart contract. BURN ignores `data` for processing token transfer, but uses `data` in Transaction Hash deriving, and signing process.

- **ERC20 token:**
  - to: ERC20 token smart contract address
  - value: 0
  - data: RLP encoded value of an array data ['a9059cbb', to address, amount]

Note: On Ethereum, a sender can transfer ETH at the same time by specifying `value`. BURN ignores `data` for processing token transfer, but uses `value` in Transaction Hash deriving, and signing process.

Note: 'a9059cbb' is the hash value representing 'transfer(address,uin256)'

`burnweb` and `BURN REST API` take care of the `to` and `value` keeping compatibility with the above mechanism.

- **Native token:**
  - to: The address to which a sender transfer the token.
  - value: The amount of the native token to transfer.
  - data: '0x'

Note: The native token is configured in .env, and internally it's represented as tokenId = '0x0000000000000000000000000000000000000000'

- **Other tokens:**
  - to: tokenId
  - value: 0

  - data: RLP encoded value of an array data [‘a9059cbb’, to address, amount]

Transactions to Create Token, Create BURN KVS, Set/Delete Key-Value use `to` and `value` as follows:

- to: Targetted token’s tokenId or KVS’s storeId
- value: 0
- data: RLP encoded value of parameters array according to each action

2.4 chainId

Unique number (positive integer) for each blockchain. Ethereum Mainnet uses a chainId `1`. There are some well-known chainIds:

Table 1: Chain IDs

chainId	Network Name
1	Mainnet
3	Ropsten
4	Rinkeby
42	Kovan
61	Ethereum Classic Mainnet

In BURN, configure chainId other than above listed.

CHAPTER

THREE

TRANSACTION FEE

3.1 Gas for each operation

We define gas fee in static native token amount for each operation. We don't use Ethereum-like gasPrice and gasLimit style.

- GAS_CREATE_TOKEN
- GAS_MINT_TOKEN
- GAS_BURN_TOKEN
- GAS_TRANSFER_TOKEN
- GAS_CREATE_KVS
- GAS_SET_KEY_VALUE
- GAS_DELETE_KEY_VALUE

These values are configured in .env. At the first launch of the BURN network in the initialization process, these values are loaded from .env, and persisted. Collected gas will be transferred into `NETWORK_OWNER` also configured in .env.

3.2 Token related gas

3.2.1 Create/Mint/Burn Token

- When a new token is created, the sender of the transaction is charged `GAS_CREATE_TOKEN` amount of native token as a fee.
- The sender created the token is set as `Token Owner` of the created token.
- For each token, we can define `fee_token_id`, `fee`, and `fee_rate`.
- If the token is configured `mintable`, the `Token Owner` can mint (issue) token. The `Token Owner` is charged `GAS_MINT_TOKEN` as a gas fee.
- If the token is configured `burnable`, the `Token Owner` can burn token. The `Token Owner` is charged `GAS_BURN_TOKEN` as a gas fee.

3.2.2 Transfer Token

- If `fee_token_id` is not set or set to '0x0000000000000000000000000000000000000000' - The token sender pays gas fee, with amount GAS_TRANSFER_TOKEN to NETWORK_OWNER.
- If `fee_token_id` is set other than the native token - The token sender pays gas in `fee_token_id` token to `Token Owner`, with amount max(transfer amount * `fee_rate` / 100000, `fee`). Also, `Token Owner` pays native token with amount `GAS_TRANFER_TOKEN` to `NETWORK_OWNER`.

3.2.3 KVS related gas

- When a new KVS (Key-Value Store) is created, the sender of the transaction is charged GAS_CREATE_STORE amount of native token as a fee.
- The sender created the KVS is set as `Store Owner` of the created KVS.
- For each KVS, we can define `fee_token_id`, `fee`, and `fee_rate`.
- If `fee_token_id` is not set or set to '0x0000000000000000000000000000000000000000' - The Set/Delete transaction sender pays gas fee, with amount GAS_SET/DELETE_TOKEN to NETWORK_OWNER.
- If `fee_token_id` is set other than the native token - The Set/Delete transaction sender pays gas in `fee_token_id` token to `Store Owner`, with amount max(transfer amount * `fee_rate` / 100000, `fee`). Also, `Token Owner` pays native token with amount GAS_SET/DELETE_TOKEN to NETWORK_OWNER.

CHAPTER

FOUR

BURNWEB

This is the main class of the burnweb library. This page shows static methods.

```
var BurnWeb = require('burnweb');
```

4.1 generateAccount

```
const account = BurnWeb.generateAccount([entropy]);
```

Returns random private key and account address.

4.1.1 Parameters

1. `String` - (Optional) Random seed string to add entropy.

4.1.2 Returns

`Object`- A pair of private key and account address.

- `address` - `String`: The derived account address from the private key.
- `privateKey` - `String`: The specified private key.

4.1.3 Example

```
BurnWeb.generateAccount()
> {
    address: '0x61ecf32a6286f91f765645c4bf2d5dde7775b907',
    privateKey: '0x93cd9c0e9ef8ac5eac8552c1b2379e17b1aa569e7f5a50bae9e80d07999abd15'
}
```

4.2 privateKeyToAccount

```
const account = BurnWeb.privateKeyToAccount(privateKey);
```

Returns a derived account address from the specified private key.

4.2.1 Parameters

1. `String` - The private key.

4.2.2 Returns

`Object` - The account address derived from the private key.

- `address` - `String`: The derived account address from the private key.
- `privateKey` - `String`: The specified private key.

4.2.3 Example

```
BurnWeb.privateKeyToAccount(
↪'0x93cd9c0e9ef8ac5eac8552c1b2379e17b1aa569e7f5a50bae9e80d07999abd15')
> {
    address: '0x61ecf32a6286f91f765645c4bf2d5dde7775b907',
    privateKey: '0x93cd9c0e9ef8ac5eac8552c1b2379e17b1aa569e7f5a50bae9e80d07999abd15' }
}
```

CHAPTER

FIVE

BURNWEB NETWORK

This is the main class of the burnweb library.

```
var BurnWeb = require('burnweb');

// Set BURN endpoint URL to initialize
var burnweb = new BurnWeb('http://localhost:8545');

// or, specify private key at the same time
var burnweb = new BurnWeb('http://localhost:8545', privateKey);
```

5.1 getBlockNumber

```
burnweb.getBlockNumber([callback])
```

Returns the current block number.

5.1.1 Returns

`Promise` returns `Number` - The latest block number.

5.1.2 Example

```
burnweb.getBlockNumber()
.then(console.log);
> 1234
```

5.2 getBlock

```
burnweb.getBlock(blockNumber [, callback])
```

Returns a block information for the specified block number.

5.2.1 Parameters

1. `Number|BN|BigNumber` - The block number.
2. `Function` - (optional) Optional callback.

5.2.2 Returns

`Promise` returns `Object` - The block object:

- `block_number` - `Number`: The block number. `null` if a pending block.
- `checkpoint_number` - `Number`: The block number. `null` if a pending block.
- `parent_hash` 32 Bytes - `String`: Hash of the parent block.
- `tx_root_hash` 32 Bytes - `String`: Merkle hash of the transactions.
- `block_hash` 32 Bytes - `String`: Hash of the current block.
- `created` - `DateTime`: The block created date time formatted in 'YYYY-MM-DD HH:mm:ss'

5.2.3 Example

```
burnweb.getBlock(1234)
.then(console.log);
> {
    "block_number": 1234,
    "checkpoint_number": 123,
    "parent_hash": "0x10dd40dd6d96d36897d01c38a7047f17920f8eefe05af01800ba07c0be269099
↪",
    "tx_root_hash":
↪"0x0bf84e5a9a443c22eb4a7d1446213ed67da18b64a1fbcad3f50e699f4d2233cb",
    "block_hash": "0x7901d33b19010f009bb1fb8c31d7f522ebced81188bf9e4f3255e8c5c47cacbc
↪",
    "created": "2020-08-02 10:02:11"
}
```

5.3 getBalance

```
burnweb.getBalance(address [, callback])
```

Get the native token balance of a specified address.

5.3.1 Parameters

1. `String` - The address to get the balance of native token.
2. `Function` - (optional) Optional callback.

5.3.2 Returns

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

5.3.3 Example

```
burnweb.getBalance("0xa1d8ba23b27c334b01b6260a2eb6d767fa035cb2")
.then(console.log);
> "1000000000000"
```

5.4 getTransaction

```
burnweb.getTransaction(tx_hash [, callback])
```

Returns a transaction details for a specified tx hash.

5.4.1 Parameters

1. `String` - The transaction hash.
2. `Function` - (optional) Optional callback.

5.4.2 Returns

`Promise` returns `Object` - The transaction object:

- `tx_id` 32 Bytes - `String`: Hash of the transaction.
- `block_number` - `Number`: The block number where the transaction is included.
- `nonce` - `Number`: Unique number for the transaction.
- `token_id` - `String`: If token transfer transaction, the token id of the transferred token.
- `source` - `String`: If token transfer transaction, the addresss of the token sender.
- `target` - `String`: If token transfer transaction, the addresss of the token receiver.

- `amount` - `Number`: If token transfer transaction, the amount of token transferred.
- `fee` - `Number`: Transaction fee deducted. See *fee* for more details.
- `data` - `String`: RLP endoded parameters. See *data* for more details.
- `signature` - `String`: Transaction signature. See *signature* for more details.
- `created` - `DateTime`: The transaction created date time formatted in 'YYYY-MM-DD HH:mm:ss'

5.4.3 Example

```
burnweb.getTransaction(
↪'0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b$234')
.then(console.log);
> {
    "tx_id": "0x4c095d0b9746625158ac68352f32f4af69d8b35dbb54fdac03c487b887de32ee",
    "block_number": 26,
    "nonce": 38706715,
    "token_id": "0xb02d1e0a6680bb1f236acddc4d1797dad9e0041e",
    "source": "0xa2710da45f1343c9ee2f88d0e64ea0c8aaadfeff",
    "target": "0x50B192630d0685570e1DBECAF045011bec139b14",
    "amount": 1000,
    "fee": 0,
    "data": null,
    "signature":
↪"25a0873e52000b760903e922fe3dde6dd895b1e88afb4f379633a4d74351b5ddba183fba1ccbdebf46ab0fb7f8632a944
↪",
    "created": "2020-09-24 03:31:57",
}
```

CHAPTER

SIX

BURNWEB TOKEN

6.1 getToken

```
burnweb.getToken(tokenId [, callback])
```

Get the token information.

6.1.1 Parameters

1. `String` - The token ID.
2. `Function` - (optional) Optional callback.

6.1.2 Returns

`Promise` returns `Object` - The token object.

- `token_id` - `String`: The token ID.
- `owner` - `String`: The token owner.
- `name` - `String`: Token name.
- `symbol` - `String`: Token symbol.
- `decimals` - `Number`: Token's decimal place.
- `total_supply` - `String`: Current total supply of the token.
- `mintable` - `Number`: 0 or 1 (0: not mintable, 1: mintable)
- `burnable` - `Number`: 0 or 1 (0: not burnable, 1: burnable)
- `icon` - `String`: URL to the token icon image.
- `tx_fee` - `String`: See details at *Transaction Fee*
- `tx_fee_rate` - `String`: See details at *Transaction Fee*
- `created` - `DateTime`: The token created date time formatted in 'YYYY-MM-DD HH:mm:ss'

6.1.3 Example

```
burnweb.getBlock(1234)
.then(console.log);
> {
    "token_id": "0x004b1d45cbd495aae40bc921e318d27fffb1d357",
    "owner": "0x3a762d996bbb3633c653e1dcb0201663874dc9e2",
    "name": "Name",
    "symbol": "USDN",
    "decimals": 6,
    "total_supply": "20000000000000",
    "mintable": 1,
    "burnable": 0,
    "icon": "https://s3.aws.com/11980234/219315.png",
    "tx_fee": "0",
    "tx_fee_rate": "0",
    "created": "2020-11-16 11:17:50"
}
```

6.2 getBalanceOf

```
burnweb.getBalance(tokenId, address [, callback])
```

Get the token balance of a specified address.

6.2.1 Parameters

1. `String` - The token ID.
2. `String` - The address to get the balance of the token.
3. `Function` - (optional) Optional callback.

6.2.2 Returns

`Promise` returns `String` - The current balance of the token for the given address

6.2.3 Example

```
burnweb.getBalanceOf("0x003d6e501a19921a63a9046f5da10675bc0965b2",
↪"0xa1d8ba23b27c334b01b6260a2eb6d767fa035cb2")
.then(console.log);
> "1000000000000"
```

6.3 createToken

```
burnweb.createToken(
    name,
    symbol,
    decimals,
    totalSupply,
    feeToken,
    txFee,
    txFeeRate,
    icon,
    mintable,
    burnable
    [, callback]
)
```

Create a new token.

6.3.1 Parameters

1. `String` - Token name.
2. `String` - Token symbol.
3. `Number` - Token's decimal place.
4. `String` - Initial total supply of the token.
5. `Number` - 0 or 1 (0: Transaction fee is charged in the native token, 1: Transaction fee is charged in the token itself)
6. `String` - Minimum transaction fee for token transfer. See detials at *Transaction Fee*
7. `String` - Transaction fee in rate to transferred token amount. See detials at *Transaction Fee*
8. `String` - URL to the token icon image.
9. `Number` - 0 or 1 (0: not mintable, 1: mintable)
10. `Number` - 0 or 1 (0: not burnable, 1: burnable)
11. `Function` - (optional) Optional callback.

6.3.2 Returns

`Promise` returns `Object` - Transaction hash, and the created token ID.

- `txHash` - `Number`: Transaction hash.
- `tokenId` - `Number`: The created token ID.

6.3.3 Example

```
burnweb.getBalanceOf(
    "Alpha USD",
    "USDA",
    18,
    "200000000000000000000000",
    "0x0000000000000000000000000000000000000000",
    "0",
    "0",
    "icon": "https://burn-network.io/images/udsa.png",
    "mintable": 1,
    "burnable": 0
).then(console.log);
> { txHash: "0xaba239fc212acfd893282e8bca573c72d5b5c1cdf99321700f38147510a8fb6d",
↪"0x97BcC3F68DBcAe2382308C46A59b76fA2f8116f8" }
```

6.4 transferToken

```
burnweb.transferToken(tokenId, to, amount [, callback])
```

Transfer token to another address.

6.4.1 Parameters

1. `String` - The token ID.
2. `String` - Transfer the token to this address.
3. `String` - The token amount to transfer.
4. `Function` - (optional) Optional callback.

6.4.2 Returns

`Promise` returns `String` - Transaction hash

6.4.3 Example

```
burnweb.transferToken("0x97bcc3f68dbcae2382308c46a59b76fa2f8116f8",
↪"0xa2710da45f1343c9ee2f88d0e64ea0c8aaadfeff", "20000")
.then(console.log);
> "0xaba239fc212acfd893282e8bca573c72d5b5c1cdf99321700f38147510a8fb6d"
```

6.5 issueToken

```
burnweb.issueToken(tokenId, to, amount)
```

Issue (mint) token to a specified address.

6.5.1 Parameters

1. `String` - The token ID.
2. `String` - The address to which the token is issued.
3. `String` - Token amount to issue.
4. `Function` - (optional) Optional callback.

6.5.2 Returns

`Promise` returns `String` - Transaction hash

6.5.3 Example

```
burnweb.issueToken("0x97BcC3F68DBcAe2382308C46A59b76fA2f8116f8",
↪"0xa2710da45f1343c9ee2f88d0e64ea0c8aaadfeff", "100000000000000000000000000")
.then(console.log);
> "0x84dd7f59a662fa159808881a60a669319fd0366006b8c8c3d293860abc4b46da"
```

6.6 burnToken

```
burnweb.burnToken(tokenId, amount)
```

Burn token balance.

6.6.1 Parameters

1. `String` - The token ID.
2. `String` - Token amount to burn.
3. `Function` - (optional) Optional callback.

6.6.2 Returns

`Promise` returns `String` - Transaction hash

6.6.3 Example

```
burnweb.burnToken("0x97BcC3F68DBcAe2382308C46A59b76fA2f8116f8", "100000000000000000000
↪")
.then(console.log);
> "0xe0509cc9e9569a693e38246cffaeb3997d44167c9cc2500088505cd68abe9d23"
```

6.7 listTokenTransactions

```
burnweb.listTokenTransactions(tokenId, from, to, start, end [, callback])
```

List transactions for specified token.

6.7.1 Parameters

1. `String` - The token ID.
2. `String` - (optional) Token sender
3. `String` - (optional) Token receiver
4. `String` - DateTime range start (e.g. "2021-02-15 07:52:37")
5. `String` - DateTime range end (e.g. "2025-02-15 01:00:00")
6. `Function` - (optional) Optional callback.

6.7.2 Returns

`Promise` returns `Object` - The list of transaction object.

- `tx_id` 32 Bytes - `String`: Hash of the transaction.
- `block_number` - `Number`: The block number where the transaction is included.
- `nonce` - `Number`: Unique number for the transaction.
- `token_id` - `String`: If token transfer transaction, the token id of the transferred token.
- `source` - `String`: If token transfer transaction, the addresss of the token sender.
- `target` - `String`: If token transfer transaction, the addresss of the token receiver.
- `amount` - `Number`: If token transfer transaction, the amount of token transferred.
- `fee` - `Number`: Transaction fee deducted. See *fee* for more details.
- `data` - `String`: RLP endoded parameters. See *data* for more details.
- `signature` - `String`: Transaction signature. See *signature* for more details.
- `created` - `DateTime`: The transaction created date time formatted in 'YYYY-MM-DD HH:mm:ss'

6.7.3 Example

```
burnweb.listTokenTransactions("0x0000000000000000000000000000000000000000", undefined,
↪ undefined, "2021-02-15 00:0:00", "9999-01-01 00:00:00")
.then(console.log);
> [
    {
      "tx_id": "0x335341f1cba6eb17b7c6cbf76d59db5a8ba4f936cd88977892c704b089f0b91a",
      "block_number": "8898401",
      "nonce": "388348616525703",
      "token_id": "0x0000000000000000000000000000000000000000",
      "source": "0x4bea9f4ebba2c63289fb257ed58df1e0e572b1e4",
      "target": "0x93106e4f822ec69776f8fa8dfa514701308cd510",
      "amount": "1100000000000000000",
      "fee": "100",
      "data": "",
      "signature":
↪"8422d2f46d65626f0be85788d8595135fbf4af048b1a77fa00495ff9ba17b6bc94ee66066ed5b64217c57a727f3ce833a3
↪",
      "created": "2021-02-15 07:52:46"
    }
  ]
```

CHAPTER

SEVEN

BURNWEB KVS

7.1 createStore

```
burnweb.createToken(storeName [, callback])
```

Create a new Key-Value Store.

7.1.1 Parameters

1. `String` - Key-Value Store name.
2. `Function` - (optional) Optional callback.

7.1.2 Returns

`Promise` returns `Object` - Transaction hash, and the created Key-Value Store ID.

- `txHash` - `Number`: Transaction hash.
- `storeId` - `Number`: The created Key-Value Store ID.

7.1.3 Example

```
burnweb.createStore("KVS01").then(console.log);
> { txHash: "0xaba239fc212acfd893282e8bca573c72d5b5c1cdf99321700f38147510a8fb6d",
↪"0x0c0573302634DC279302f9dD65241C9825FFAC98" }
```

7.2 setKeyValue

```
burnweb.setKeyValue(storeId, collection, key, value)
```

Set key-value pair for specified Key-Value Store ID and collection name.

7.2.1 Parameters

1. `String` - The Key-Value Store ID.
2. `String` - Collection name.
3. `String` - Key string.
4. `String` - Value string.
5. `Function` - (optional) Optional callback.

7.2.2 Returns

`Promise` returns `String` - Transaction hash

7.2.3 Example

```
burnweb.setKeyValue("0x0c0573302634DC279302f9dD65241C9825FFAC98", "Collection01",
↪"Key001", "Value001")
.then(console.log);
> "0xeff8d7b1a4aee0d88be01c58d0d2d1a5a2d12618a27cfcfd76ebd382fdc1bf8c"
```

7.3 deleteKeyValue

```
burnweb.deleteKeyValue(storeId, collection, key)
```

Delete key-value pair for specified Key-Value Store ID, collection name, and key string.

7.3.1 Parameters

1. `String` - The Key-Value Store ID.
2. `String` - Collection name.
3. `String` - Key string.
4. `Function` - (optional) Optional callback.

7.3.2 Returns

`Promise` returns `String` - Transaction hash

7.3.3 Example

```
burnweb.deleteKeyValue("0x0c0573302634DC279302f9dD65241C9825FFAC98", "Collection01",
↪"Key001")
.then(console.log);
> "0x9695bbd8b1650fa4b5c1916f212076ae1820ad56a34778aa57c669c777460309"
```

7.4 getKeyValue

```
burnweb.getKeyValue(storeId, collection, key)
```

Get key-value pair for specified Key-Value Store ID, collection name, and key string.

7.4.1 Parameters

1. `String` - The Key-Value Store ID.
2. `String` - Collection name.
3. `String` - Key string.
4. `Function` - (optional) Optional callback.

7.4.2 Returns

`Promise` returns `Object` - The Key-Value object.

- `store_id` - `String`: The Key-Value Store ID.
- `collection` - `String`: Collection name.
- `key` - `String`: Key string.
- `value` - `String`: Value string.

7.4.3 Example

```
burnweb.getKeyValue("0x0c0573302634DC279302f9dD65241C9825FFAC98", "Collection01",
↪"Key001")
.then(console.log);
> {
    "store_id": "0x0c0573302634DC279302f9dD65241C9825FFAC98",
    "collection": "Collection01",
    "key": "Key001",
    "value": "Value001"
}
```