bubi Documentation

Release 1.0.0

wilson

Sep 10, 2019

Contents

Contents:

CHAPTER 1

BUMO Keypair Guide

1.1 Overview

This document describes in detail the process of generating Keypairs (public and private key pairs) and how to generate an address and sign a transaction based on keypairs. It introduces two interface methods and related processes for executing the transaction call. It provides reference information for ProtoBuf data structures. Finally, it illustrates two methods to submit transactions by showing how to generate transaction_blob with interface call and how to generate transaction_blob by yourself.

1.2 Terminology

This section gives details about the terms used in this document.

Keypair

In the BUMO project, the keypair is the interface that generates the public key, private key, address, and signature. Only the `ED25519` signature algorithm is supported during the signing process.

Private Key

The private key is a string generated by the algorithm. The private key is a prerequisite for generating the public key and the address, and is also the basic element for completing the signature. The private key cannot be changed after it is generated. Once it is lost, it cannot be retrieved, so it needs to be kept safely.

Public Key

The public key is a string generated based on the private key. It can verify the string encrypted by the private key. It does not expose the private key when transmitted between networks. It is also a necessary condition for generating an address.

Address

The address is a string generated upon the public key. Similar to real-life addresses, contacts cannot be found without an address, so transactions cannot be completed.

Signature

The Signature refers to the process of encrypting and confirming transaction data by algorithm and private key and obtaining signature data. The user can verify the integrity and correctness of the transaction data through the signature data.

Transaction

All operations that modify blockchain data in BUMO are called transactions, such as issuing assets, transferring assets, sending BUs, creating accounts, setting metadata and setting permissions, etc.

Transaction Blob

The Transaction Blob is a hexadecimal string obtained by serializing a transaction object. Transaction serialization refers to the process of converting the state information of a transaction object into a string that can be stored and transmitted through the ProtoBuf data structure.

Raw Private Key

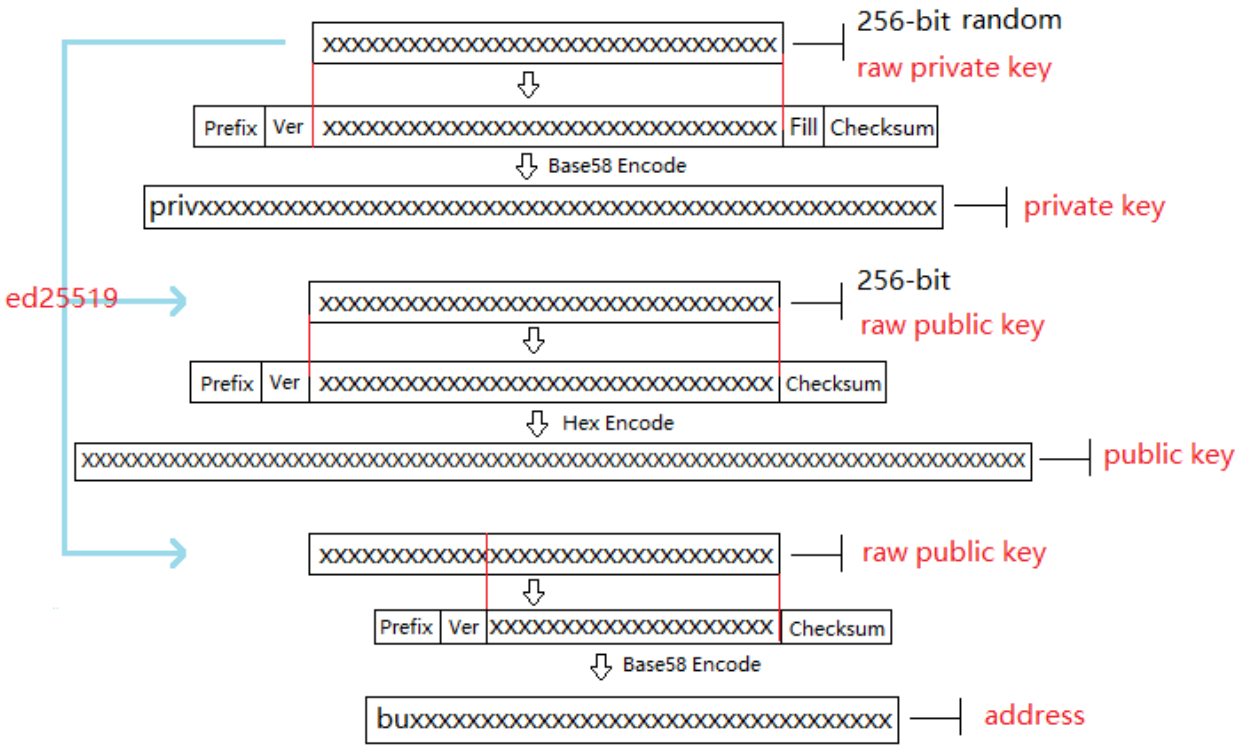
The Raw Private Key is a byte array obtained by a random algorithm. The Raw Private Key is a prerequisite for generating a private key.

Raw Public Key

The Raw Public Key is a byte array generated by processing the raw private key with the `ED25519` algorithm. The Raw Public Key is a prerequisite for generating a public key.

1.3 Schematic Diagram

The following diagram illustrates how the private, public keys and address are generated.



1.4 Generating Private Keys

Generating a private key requires multiple algorithms such as a random algorithm and `SHA256`. Generating a private key includes the following steps:

1. Generate a 256-bit random number (a private key in the mathematical sense) with a random algorithm and get a byte array, the raw private key, as shown below:

```
[17,236,24,183,207,250,207,180,108,87,224,39,189,99,246,85,138,120,236,78,228,233,41,
↪192,124,109,156,104,235,66,194,24]
```

2. Add a 3-byte prefix in the raw private key, and then add a 1-byte version number to get a new byte array, as shown below:

```
[218,55,159,1,17,236,24,183,207,250,207,180,108,87,224,39,189,99,246,85,138,120,236,
↪78,228,233,41,192,124,109,156,104,235,66,194,24]
```

Note: For the Prefix, Version and Checksum, please refer to Table 1.

3. Perform `SHA256` calculations twice on the byte array obtained in Step 2. Take the first 4 bytes of the operation result as the byte array of the Checksum, as shown below:

```
[30,19,80,117]
```

4. Combine the byte array in Step 2 and the checksum byte array in Step 3 in order, resulting in a new byte array, as shown below:

```
[218,55,159,1,17,236,24,183,207,250,207,180,108,87,224,39,189,99,246,85,138,120,236,
↪78,228,233,41,192,124,109,156,104,235,66,194,24,30,19,80,117]
```

5. Encode the byte array generated in Step 4 with Base58, and get the string starting with priv, namely the private key, as shown below:

```
privbsGZFUoRv8aXZbSGd3bwzZWFn3L5QKq74RXAQYcmfXhhZ54CLr9z
```

Note: Now the private key is generated.

Table 1

Name	Data	Length
Prefix	0xDA 0x37 0x9F	3 bytes
Version	0x01	1 byte
Checksum	After performing `SHA256` calculation twice on the byte array obtained in Step 2,take the first 4 bytes of the operation result	4 bytes

This table illustrates the Prefix, Version and Checksum used in generating the private key.

1.5 Generating Public Keys

The public key can be generated with the `ED25519` algorithm after the private key is generated. Generating a public key includes the following steps:

1. Generate a 32-bit byte array (raw public key) by processing the raw private key with the `ED25519` algorithm. For example, the raw public key of the private key `privbsGZFUoRv8aXZbSGd3bwzZWFn3L5QKq74RXAQYcmfXhhZ54CLr9z` is shown below:

```
[21,118,76,208,23,224,218,117,50,113,250,38,205,82,148,81,162,27,130,83,208,1,240,212,
↪54,18,225,158,198,50,87,10]
```

2. Add a 1-byte prefix in the raw public key, and then add a 1-byte version number to get a new byte array, as shown below:

```
[176,1,21,118,76,208,23,224,218,117,50,113,250,38,205,82,148,81,162,27,130,83,208,1,
↪240,212,54,18,225,158,198,50,87,10]
```

Note: For the Prefix, Version and Checksum, please refer to Table 2.

3. Perform `SHA256` calculation twice on the byte array in Step 2. Take the first 4 bytes of the operation result as the byte array of the Checksum, as shown below:

```
[116,171,22,107]
```

4. Combine the byte array in Step 2 and the checksum byte array in Step 3 in order, resulting in a new byte array, as shown below:

```
[176,1,21,118,76,208,23,224,218,117,50,113,250,38,205,82,148,81,162,27,130,83,208,1,
↪240,212,54,18,225,158,198,50,87,10,116,171,22,107]
```

5. Encode the byte array in Step 4 into hexadecimal and get a hexadecimal string, namely the public key, as shown below:

```
b00115764cd017e0da753271fa26cd529451a21b8253d001f0d43612e19ec632570a74ab166b
```

Note: Now the public key is generated.

Table 2

Name	Data	Length
Prefix	0xB0	1 Byte
Version	0x01	1 Byte
Checksum	After performing `SHA256` calculation twice on the byte array obtained in Step 2, take the first 4 bytes of the operation result	4 Bytes

This table illustrates the Prefix, Version and Checksum used in generating the public key.

1.6 Generating Addresses

The address can be further generated by an algorithm after generating the private key and the public key. Generating an address includes the following steps:

1. Generate a 32-bit byte array (raw public key) by processing the raw private key with the `ED25519` algorithm. For example, the raw public key of the private key `privbsGZFUoRv8aXZbSGd3bwzZWFn3L5QKq74RXAQYcmfXhhZ54CLr9z` is shown below:

```
[21,118,76,208,23,224,218,117,50,113,250,38,205,82,148,81,162,27,130,83,208,1,240,212,
↪54,18,225,158,198,50,87,10]
```

2. Perform `SHA256` calculation twice on the raw public key and take the last 20 bytes of the operation result as the byte array, as shown below:

```
[173,148,59,51,183,193,55,160,1,133,247,80,65,13,67,190,164,114,18,220]
```

3. Add a 2-byte prefix in the byte array generated in Step 2, and then add a 1-byte version number to get a new byte array, as shown below:

```
[1,86,1,173,148,59,51,183,193,55,160,1,133,247,80,65,13,67,190,164,114,18,220]
```

Note: For the Prefix, Version and Checksum, please refer to Table 3.

4. Perform `SHA256` calculation twice on the byte array in Step 3. Take the first 4 bytes of the operation result as the byte array of the Checksum, as shown below:

```
[167,127,34,35]
```

5. Combine the byte array in Step 3 and the Checksum byte array in Step 4 in order, resulting in a new byte array, as shown below:

```
[1,86,1,173,148,59,51,183,193,55,160,1,133,247,80,65,13,67,190,164,114,18,220,167,127,
↪34,35]
```

6. Encode the byte array generated in Step 5 with Base58, and get the string starting with bu, namely the address, as shown below:

```
buQmWJrdYJP5CPKTbkQUqscwvTGaU44dord8
```

Note: Now the address is generated.

Table 3

Name	Data	Length
Prefix	0x01 0x56	2 Bytes
Version	0x01	1 Byte
PublicKey	Take the last 20 bytes in raw public key	20 Bytes
Checksum	After performing `SHA256` calculation twice on the byte array obtained in step 3, take the first 4 bytes of the operation result	4 Bytes

This table illustrates the Prefix, Version and Checksum used in generating the address.

1.7 Signing Transactions

Sign the pending transaction (the byte array obtained by the inverse hexadecimal encoding of the transaction_blob) with the `ED25519` algorithm and the private key, and perform hexadecimal conversion to get sign_data, the signature string.

The following example shows how to sign the transaction_blob with `ED25519` and the private key.

The private key:

```
b00115764cd017e0da753271fa26cd529451a21b8253d001f0d43612e19ec632570a74ab166b
```

The transaction_blob:

```
0A24627551566B5555424B70444B526D48595777314D553855376E676F5165686E6F31363569109F0818C0843D20E8073214
```

After signing the transaction_blob with the signature interface of `ED25519` and performing hexadecimal conversion, the resulting sign_data is:

```
a46ee590a84abdeb8cc38ade1ae8e8a2c71bb69bdc4cd7dc0de1b74b37e2cbd1696229687f80dff4276b1a3dd3f95a9bc1d5
```

1.8 Methods of Submitting Transactions

There are two methods of calling the interface to execute transactions: Generating Transaction_blobs by Calling the Interface and Generating Transaction_blobs by Yourself.

1.8.1 Generating Transaction_blobs by Calling the Interface

> **Attention:** As the transaction_blob is likely to be intercepted and tampered with, it is not recommended to generate transaction_blobs in this way.

If you need to call the interface to generate transaction_blobs, sign and submit transactions, please refer to the BUMO development documentation at the following address:

https://github.com/bumoproject/bumo/blob/master/docs/develop.md

Calling the interface to generate a transaction_blob includes the following steps:

1. Call the getAccount interface to get the nonce value of the account that is to initiate a transaction. The code is shown below:

```
HTTP GET host:port/getAccount?address=account address
```

2. Populate the json data as needed and complete filling the transaction data. The format is shown below:

```
{
"source_address":"xxxxxxxxxx", //The source transaction account, the originator of␣
↪the transaction
"nonce":2, //Nonce value
```

(continues on next page)

(continued from previous page)

```
"ceil_ledger_seq": 0, //Optional
"fee_limit":1000, //Fee paid in transaction
"gas_price": 1000, //Gas price (Not less than the configured value)
"metadata":"0123456789abcdef", //Optional, metadata for the transaction given by␣
↪users, in hexadecimal format
"operations":[
{
//Populate according to specific operations
},
{
//Populate according to specific operations
}
......
]
}
```

Note: The nonce value needs to be incremented by 1 based on the value obtained in Step 1.

3. By calling the getTransactionBlob interface, the json data generated in Step 2 is passed as a parameter, and a transaction hash and a transaction_blob are obtained to implement transaction serialization. The format is shown below:

```
{
"error_code": 0,
"error_desc": "",
"result": {
"hash": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", //Transaction hash
"transaction_blob": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" //The hexadecimal␣
↪representation after the transaction is serialized
}
}
```

4. Sign the transaction and populate the transaction data. Sign the transaction_blob according to the previously generated private key, and then populate the json data of the submitted transaction. The format is shown below:

```
{
"items" : [{
"transaction_blob" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", //The␣
↪hexadecimal representation after the transaction is serialized
"signatures" : [{//The first signature
"sign_data" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", //Signature data
"public_key" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" //Public key
}, {//The second signature
"sign_data" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", //Signature data
"public_key" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" //Public key
}
]
}
]
}
```

5. By calling the submitTransaction interface, the json data generated in Step 4 is passed as a parameter, the response result is obtained and transaction submission is completed. The format of the response result is shown below:

```
{
"results": [
{
"error_code": 0,
"error_desc": "",
"hash": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" //Transaction hash
}
],
"success_count": 1
}
```

1.8.2 Generating Transaction_blobs by Yourself

Generating the transaction_blob by yourself, signing, and submitting the transaction include the following steps:

1. Call the getAccount interface to get the nonce value of the account that is to initiate a transaction. The code is shown below:

```
HTTP GET host:port/getAccount?address=account address
```

2. Populate the transaction object (Transaction) of the protocol buffer and serialize it to get the transaction_blob. For details of the specific transaction data structure, please refer to *ProtoBuf Data Structure*.

3. Sign the transaction and populate the transaction data. Generate a public key based on the private key, sign the transaction _blob with the private key, and then populate the json data of the submitted transaction. The format is shown below:

```
{
"items" : [{
"transaction_blob" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", //The␣
↪hexadecimal representation after the transaction is serialized
"signatures" : [{//The first signature
"sign_data" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", //Signature data
"public_key" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" //Public key
}, {//The second signature
"sign_data" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", //Signature data
"public_key" : "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" //Public key
}
]
}
]
}
```

4. By calling the submitTransaction interface, the json data generated in Step 3 is passed as a parameter to complete the transaction submission. The response result format is shown below:

```
{
"results": [
{
"error_code": 0,
"error_desc": "",
"hash": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" //Transaction hash
}
],
"success_count": 1
}
```

1.9 ProtoBuf Data Structure

Protocol Buffer (ProtoBuf) is a lightweight and efficient structured data storage format that can be used for serializing structured data. It is ideal for data storage or RPC data exchange formats. It can be used in communication protocols, data storage and other fields of language-independent, platform-independent, scalable serialized structured data formats. Currently the APIs in `C++`, `Java`, and `Python` are available.

For more information aobut ProtoBuf, please refer to the following link:

https://developers.google.com/protocol-buffers/docs/overview

Now, we will introduce the data structure details of Protocol Buffer, and provide the file and simple test program for the protocol buffer of various languages generated by the script.

1.9.1 Data Structure

The following section describes the various ProtoBuf data structures that might be used in transactions and their uses for your reference.

Transaction

This data structure is for complete transactions.

```
message Transaction {
enum Limit{
UNKNOWN = 0;
OPERATIONS = 1000;
};
string source_address = 1; // Account address of the transaction initiator
int64 nonce = 2; // Transaction sequence number
int64 fee_limit = 3; // The transaction fee, by default is 1000Gas; the unit is MO, 1␣
↪BU = 10^8 MO
int64 gas_price = 4; // The packaging fee of transactions, by default is 1000; the␣
↪unit is MO,1 BU = 10^8 MO
int64 ceil_ledger_seq = 5; // Block bound
bytes metadata = 6; // Transaction metadata
repeated Operation operations = 7; // Operation list
}
```

Operation

This data structure is for operations in transactions.

```
message Operation {
enum Type {
UNKNOWN = 0;
CREATE_ACCOUNT = 1;
ISSUE_ASSET = 2;
PAY_ASSE = 3;
SET_METADATA = 4;
SET_SIGNER_WEIGHT = 5;
SET_THRESHOLD = 6;
PAY_COIN = 7;
LOG = 8;
SET_PRIVILEGE = 9;
};
Type type = 1; // Operation type
```

(continues on next page)

(continued from previous page)

```
string source_address = 2; // Source account address for the operation
bytes metadata = 3; // Operation metadata

OperationCreateAccount create_account = 4; // Create an account operation
OperationIssueAsset issue_asset = 5; // Issue assets operation
OperationPayAsset pay_asset = 6; // Transfer assets operation
OperationSetMetadata set_metadata = 7; // Set metadata
OperationSetSignerWeight set_signer_weight = 8; // Set privilege for signer
OperationSetThreshold set_threshold = 9; // Set transaction threshold
OperationPayCoin pay_coin = 10; // Transfer coin
OperationLog log = 11; // Record log
OperationSetPrivilege set_privilege = 12; // Set privilege
}
```

OperationCreateAccount

This data structure is for creating accounts.

```
message OperationCreateAccount{
string dest_address = 1; // Target account address to be created
Contract contract = 2; // Contract
AccountPrivilege priv = 3; // Privilege
repeated KeyPair metadatas = 4; // Additional info
int64 init_balance = 5; // Initiation balance
string init_input = 6; // Input parameter for contracts
}
```

Contract

This data structure is for setting contracts.

```
message Contract{
enum ContractType{
JAVASCRIPT = 0;
}
ContractType type = 1; // Contract type
string payload = 2; // Contract code
}
```

AccountPrivilege

This data structure is for setting account privilege.

```
message AccountPrivilege {
int64 master_weight = 1; // Account weight
repeated Signer signers = 2; // Signer weight list
AccountThreshold thresholds = 3; // Threshold
}
```

Signer

This data structure is for setting signer weight.

```
message Signer {
enum Limit{
SIGNER_NONE = 0;
SIGNER = 100;
};
```

(continues on next page)

(continued from previous page)

```
string address = 1; // Signer account address
int64 weight = 2; // Signer weight
}
```

AccountThreshold

This data structure is for setting account threshold.

```
message AccountThreshold{
int64 tx_threshold = 1; // Transaction threshold
repeated OperationTypeThreshold type_thresholds = 2; // Specify the transaction␣
↪threshold list for the operations. The threshold for the transactions with␣
↪unspecified operation is set by tx_threshold
}
```

OperationTypeThreshold

This data structure is for operation threshold of specified types.

```
message OperationTypeThreshold{
Operation.Type type = 1; // Operation type
int64 threshold = 2; // Corresponding threshold of this operation
}
```

OperationIssueAsset

This data structure is for issuing assets.

```
message OperationIssueAsset{
string code = 1; // Asset encoding to be issued
int64 amount = 2; // Asset amount to be issued
}
```

OperationPayAsset

This data structure is for transferring assets.

```
message OperationPayAsset {
string dest_address = 1; // Target account address
Asset asset = 2; // Asset
string input = 3; // Input parameter for contracts
}
```

Asset

This data structure is for asset.

```
message Asset{
AssetKey key = 1; // Asset identification
int64 amount = 2; // Asset amount
}
```

AssetKey

This data structure is for identifying the uniqueness of asset.

```
message AssetKey{
string issuer = 1; // Account address of asset issuer
```

(continues on next page)

(continued from previous page)

```
string code = 2; // Asset encoding
int32 type = 3; // Asset type(by default is 0, which indicates the amount is not␣
↪limited)
}
```

OperationSetMetadata

This data structure is for setting Metadata.

```
message OperationSetMetadata{
string key = 1; // keyword, unique
string value = 2; // Content
int64 version = 3; // Version control, optional
bool delete_flag = 4; // Whether it is deletable
}
```

OperationSetSignerWeight

This data structure is for setting signer weight.

```
message OperationSetSignerWeight{
int64 master_weight = 1; // Self weight
repeated Signer signers = 2; // Signer weight list
}
```

OperationSetThreshold

This data structure is for setting threshold.

```
message OperationSetThreshold{
int64 tx_threshold = 1; // Transaction threshold
repeated OperationTypeThreshold type_thresholds = 2; // The transaction threshold␣
↪list for specified operations. The threshold for the transactions with unspecified␣
↪operation is set by tx_threshold

}
```

OperationPayCoin

This data structure is for sending coin.

```
message OperationPayCoin{
string dest_address = 1; // Target account address
int64 amount = 2; // Coin amount
string input = 3; // Input parameter for contracts
}
```

OperationLog

This data structure is for recording log information.

```
message OperationLog{
string topic = 1; // Log theme
repeated string datas = 2; // Log content
}
```

OperationSetPrivilege

This data structure is for setting account privilege.

```
message OperationSetPrivilege{
string master_weight = 1; // Account weight
repeated Signer signers = 2; // Signer weight list
string tx_threshold = 3; // Transaction threshold
repeated OperationTypeThreshold type_thresholds = 4; // The transaction threshold␣
↪list for specified operations. The threshold for the transactions with unspecified␣
↪operation is set by tx_threshold

}
```

1.9.2 Examples

This section provides examples of proto scripts, as well as proto source code generated by `cpp`, `java`, `javascript`, `pyton`, `object-c`, and `php`. For more information, please refer to the following link:

https://github.com/bumoproject/bumo/tree/develop/src/proto

Description of the directory structure in the above link is shown below:

1. cpp: C++ source code
2. io: Java source code
3. go: Go source and test program
4. js: Javascript source code and test program
5. Python: Python source code and test program
6. ios: Object-c source code and test program
7. php: PHP source code and test program

1.10 Examples for Transaction Submission

Scenario: Account A`buQVkUUBKpDKRmHYWw1MU8U7ngoQehno165i`creates account B (Generate an address by *Generating Addresses* in Keypair).

1.10.1 Generating Transaction_blobs by Interface

Generating transaction_blobs by the interface includes the following steps:

1. Obtain the nonce value of the account to initiate a transaction by GET.

```
GET http://seed1.bumotest.io:26002/getAccount?
↪address=buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw
```

Response message:

```
{
"error_code" : 0,
"result" : {
"address" : "buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw",
"assets" : [
{
"amount" : 1000000000,
```

(continues on next page)

(continued from previous page)

```
"key" : {
"code" : "HNC",
"issuer" : "buQBjJD1BSJ7nzAbzdTenAhpFjmxRVEEtmxH"
}
}
],
"assets_hash" : "3bf279af496877a51303e91c36d42d64ba9d414de8c038719b842e6421a9dae0",
"balance" : 27034700,
"metadatas" : null,
"metadatas_hash" : "ad67d57ae19de8068dbcd47282146bd553fe9f684c57c8c114453863ee41abc3",
"nonce" : 5,
"priv" : {
"master_weight" : 1,
"thresholds" : [{
"tx_threshold" : 1
}
]
}
}
}
address: Current query account address
assets: Account asset list
assets_hash: Asset list hash
balance: Account balance
metadata: Account metadata in hexadecimal format
metadatas_hash: Transaction metadata hash
nonce: The sending transaction serial number, the nonce+1 returned by querying the␣
↪account information interface
priv: Privilege
master_weight: Current account weight
thresholds: Threshold
tx_threshold: Transaction default threshold
```

2. Complete populating the transaction data.

The account address of account B generated by *Generating Address* in Keypair is `buQoP2eRymAcUm3uvWgQ8RnjtrSnXBXfAzsV`, the populated json data is shown below:

```
{
"source_address":"buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw",
"nonce":7,
"ceil_ledger_seq": 0,
"fee_limit":1000000,
"gas_price": 1000,
"metadata":"",
"operations":[
{
"type": 1,
"create_account": {
"dest_address": "buQoP2eRymAcUm3uvWgQ8RnjtrSnXBXfAzsV",
"init_balance": 10000000,
"priv": {
"master_weight": 1,
"thresholds": {
"tx_threshold": 1
}
}
```

(continues on next page)

(continued from previous page)

```
}
}
]
}
```

Note: The nonce value is not 6, so this transaction would fail.

3. Serialize the transaction data.

```
POST http://seed1.bumotest.io:26002/getTransactionBlob
```

Request message:

```
4.1.2 populated jason data
```

Response message:

```
{
"error_code": 0,
"error_desc": "",
"result": {
"hash": "be4953bce94ecd5c5a19c7c4445d940c6a55fb56370f7f606e127776053b3b51",
"transaction_blob":
↪"0a2462755173757248314d34726a4c6b666a7a6b7852394b584a366a537532723978424e4577100718c0843d20e8073a3
↪"
}
}
```

4. Sign the transaction_blob with the private key.

Import package: import io.bumo.encryption.key.PrivateKey;

```
Private key:
privbvTuL1k8z27i9eyBrFDUvAVVCSxKeLtzjMMZEqimFwbNchnejS81

The sign_data after being signed:
9C86CE621A1C9368E93F332C55FDF423C087631B51E95381B80F81044714E3CE3DCF5E4634E5BE77B12ABD3C54554E834A306
```

5. Complete populating the transaction data.

```
{
"items" : [{
"transaction_blob" :
↪"0a2462755173757248314d34726a4c6b666a7a6b7852394b584a366a537532723978424e4577100718c0843d20e8073a3
↪",
"signatures" : [{
"sign_data" :
↪"9C86CE621A1C9368E93F332C55FDF423C087631B51E95381B80F81044714E3CE3DCF5E4634E5BE77B12ABD3C54554E834A
↪",
"public_key" :
↪"b00179b4adb1d3188aa1b98d6977a837bd4afdbb4813ac65472074fe3a491979bf256ba63895"
}
]
}
```

(continues on next page)

(continued from previous page)

```
]
}
```

6. Submit the transaction by POST.

```
POST http://seed1.bumotest.io/submitTransaction
```

Response message:

```
{
"results": [{
"error_code": 0,
"error_desc": "",
"hash": "be4953bce94ecd5c5a19c7c4445d940c6a55fb56370f7f606e127776053b3b51"
}
],
"success_count": 1
}
```

Note: "success_count":1 represents that the submission succeeded.

1.10.2 Generating Transaction_blobs by Yourself

Generating transaction_blobs by yourself (take `Java` as an example) includes the following steps:

1. Obtain the nonce value of the account that is to initiate a transaction by GET.

```
GET http://seed1.bumotest.io:26002/getAccount?
↪address=buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw
```

Response message:

```
{
"error_code" : 0,
"result" : {
"address" : "buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw",
"assets" : [
{
"amount" : 1000000000,
"key" : {
"code" : "HNC",
"issuer" : "buQBjJD1BSJ7nzAbzdTenAhpFjmxRVEEtmxH"
}
}
],
"assets_hash" : "3bf279af496877a51303e91c36d42d64ba9d414de8c038719b842e6421a9dae0",
"balance" : 27034700,
"metadatas" : null,
"metadatas_hash" : "ad67d57ae19de8068dbcd47282146bd553fe9f684c57c8c114453863ee41abc3",
"nonce" : 5,
"priv" : {
"master_weight" : 1,
"thresholds" : [{
```

(continues on next page)

(continued from previous page)

```
"tx_threshold" : 1
}
]
}
}
}
address: Current query account address
assets: Account asset list
assets_hash: Asset list hash
balance: Account balance
metadata: Account metadata in hexadecimal format
metadatas_hash: Transaction metadata hash
nonce: The sending transaction serial number, the nonce+1 returned by querying the␣
↪account information interface

priv: Privilege
master_weight: Current account weight
thresholds: Threshold
tx_threshold: Transaction default threshold
```

2. Populate the transaction data structure and generate a transaction_blob.

Import package: import io.bumo.sdk.core.extend.protobuf.Chain;

```
Chain.Transaction.Builder builder = Chain.Transaction.newBuilder();
builder.setSourceAddress("buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw");
builder.setNonce(7);

builder.setFeeLimit(1000 * 1000);
builder.setGasPrice(1000);
builder.setCeilLedgerSeq(0);
builder.setMetadata(ByteString.copyFromUtf8(""));

Chain.Operation.Builder operation = builder.addOperationsBuilder();
operation.setType(Chain.Operation.Type.CREATE_ACCOUNT);

Chain.OperationCreateAccount.Builder operationCreateAccount = Chain.
↪OperationCreateAccount.newBuilder();
operationCreateAccount.setDestAddress("buQoP2eRymAcUm3uvWgQ8RnjtrSnXBXfAzsV");
operationCreateAccount.setInitBalance(10000000);

Chain.AccountPrivilege.Builder accountPrivilegeBuilder = Chain.AccountPrivilege.
↪newBuilder();
accountPrivilegeBuilder.setMasterWeight(1);

Chain.AccountThreshold.Builder accountThresholdBuilder = Chain.AccountThreshold.
↪newBuilder();
accountThresholdBuilder.setTxThreshold(1);

accountPrivilegeBuilder.setThresholds(accountThresholdBuilder);
operationCreateAccount.setPriv(accountPrivilegeBuilder);
operation.setCreateAccount(operationCreateAccount);
String transaction_blob = HexFormat.byteToHex(builder.build().toByteArray());

The transaction_blob obtained:
0a2462755173757248314d34726a4c6b666a7a6b7852394b584a366a537532723978424e4577100718c0843d20e8073a3708
```

Note: The nonce value is not 6, so this transaction would fail.

3. Sign the transaction_blob with the private key.

Import package: import io.bumo.encryption.key.PrivateKey;

```
The private key:
privbvTuL1k8z27i9eyBrFDUvAVVCSxKeLtzjMMZEqimFwbNchnejS81

The sign_data after being signed:
9C86CE621A1C9368E93F332C55FDF423C087631B51E95381B80F81044714E3CE3DCF5E4634E5BE77B12ABD3C54554E834A306
```

4. Complete populating the transaction data.

```
{
"items" : [{
"transaction_blob" :
↪"0a2462755173757248314d34726a4c6b666a7a6b7852394b584a366a537532723978424e4577100718c0843d20e8073a37
↪",
"signatures" : [{
"sign_data" :
↪"9C86CE621A1C9368E93F332C55FDF423C087631B51E95381B80F81044714E3CE3DCF5E4634E5BE77B12ABD3C54554E834A
↪",
"public_key" :
↪"b00179b4adb1d3188aa1b98d6977a837bd4afdbb4813ac65472074fe3a491979bf256ba63895"
}
]
}
]
}
```

5. Submit the transaction by POST.

```
POST http://seed1.bumotest.io/submitTransaction
```

Response message:

```
{
"results": [{
"error_code": 0,
"error_desc": "",
"hash": "be4953bce94ecd5c5a19c7c4445d940c6a55fb56370f7f606e127776053b3b51"
}
],
"success_count": 1
}
```

Note: "success_count":1 represents that the submission succeeded.

CHAPTER 2

BUMO JAVA SDK Guide

2.1 Overview

This document details the common interfaces of the Bumo Java SDK, making it easier for developers to operate and query the BU blockchain.

2.2 Terminology

This section gives details about the terms used in this document.

Operate the BU Blockchain

Operate the BU Blockchain refers to writing data to or modifying data in the BU blockchain.

Submit Transactions

Submit Transactions refers to sending a request to write data to or modify data in the BU blockchain.

Query the BU Blockchain

Query the BU Blockchain refers to querying data in the BU blockchain.

Account Services

Account Services provide account validity checking and query interfaces.

Asset Services

Asset Services provide an asset-related query interface that follows the ATP 1.0 protocol.

Ctp10Token Services

Ctp10Token Services provide a validity check and query interfaces related to contract assets, which follows the CTP 1.0 protocol.

Contract Services

Contract Services provide a contract-related validity checking and query interfaces.

Transaction Services

Transaction Services provide a build transaction Blob interface, a signature interface, a query and a submit transaction interface.

Block Services

Block Services provide an interface to query the block.

Account Nonce Value

Account Nonce Value is used to identify the order in which the transaction is executed when the user submits the transaction.

2.3 Format of Request Parameters and Response Data

This section details the format of the request parameters and response data.

2.3.1 Request Parameters

The class name of the request parameter of the interface is composed of **Service Name+Method Name+Request**. For example, the request parameter format of the `getInfo` interface in Account Services is `AccountGetInfoRequest`.

The member of the request parameter is the member of the input parameter of each interface. For example, if the input parameter of the `getInfo` interface in Account Services is `address`, the complete structure of the request parameters of the interface is as follows:

```
Class AccountGetInfoRequest {
String address;
}
```

2.3.2 Response Data

The class name of the response data of the interface is composed of **Service Name+Method Name+Response**. For example, the response data format of the `getNonce` interface in Account Services is `AccountGetNonceResponse`.

The members of the response data include error codes, error descriptions, and return results. For example, the members of the response data of the `getInfo` interface in Assets Services are as follows:

```
Class AccountGetNonceResponse {
Integer errorCode;
String errorDesc;
AccountGetNonceResult result;
}
```

Note:

- errorCode: error code. 0 means no error, greater than 0 means there is an error
- errorDesc: error description

- result: returns the result. A structure whose class name is **Service Name+Method Name+Result**, whose members are members of the return value of each interface. For example, the result class name of the `getNonce` interface in Account Services is AccountGetNonceResult, and the member has a nonce. The complete structure is as follows:

```
Class AccountGetNonceResult {
Long nonce;
}
```

2.4 Usage

This section describes the process of using the SDK. First you need to generate the SDK implementation and then call the interface of the corresponding service. Services include account services, asset services, Ctp1.0Token services, contract services, transaction services, and block services. Interfaces are classified into public-private key address interfaces, validity check interfaces, query interfaces, and broadcast transaction-related interfaces.

2.4.1 Generating SDK Instances

The SDK instance is generated by calling the getInstance interface of the SDK. The specific call is as follows:

```
String url = "http://seed1.bumotest.io";
SDK sdk = SDK.getInstance(url);
```

2.4.2 Generating Public-Private Keys and Addresses

The public-private key address interface is used to generate the public key, private key, and address for the account on the BU blockchain. This can be achieved by directly calling the `Keypair.generator` interface. The specific call is as follows:

```
Keypair keypair = Keypair.generator();
System.out.println(keypair.getPrivateKey());
System.out.println(keypair.getPublicKey());
System.out.println(keypair.getAddress());
```

2.4.3 Checking Validity

The validity check interface is used to verify the validity of the information, and the information validity check can be achieved by directly invoking the corresponding interface. For example, to verify the validity of the account address, the specific call is as follows:

```
//
Initialize request parameters
String address = "buQemmMwmRQY1JkcU7w3nhruoX5N3j6C29uo";
AccountCheckValidRequest request = new AccountCheckValidRequest();
request.setAddress(address);

// Call the ``checkValid`` interface
AccountCheckValidResponse response =
sdk.getAccountService().checkValid(request);
```

(continues on next page)

(continued from previous page)

```
if(0 == response.getErrorCode()) {
System.out.println(response.getResult().isValid());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.4.4 Querying

The query interface is used to query data on the BU blockchain, and data query can be implemented by directly invoking the corresponding interface. For example, to query the account information, the specific call is as follows:

```
// Initialize request parameters
String accountAddress = "buQemmMwmRQY1JkcU7w3nhruo%X5N3j6C29uo";
AccountGetInfoRequest request = new AccountGetInfoRequest();
request.setAddress(accountAddress);

// Call the getInfo interface
AccountGetInfoResponse response = sdk.getAccountService().getInfo(request);
if (response.getErrorCode() == 0) {
AccountGetInfoResult result = response.getResult();
System.out.println(JSON.toJSONString(result,true));
}
else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.4.5 Broadcasting Transactions

Broadcasting transactions refers to the initiation of a transaction by means of broadcasting. The broadcast transaction consists of the following steps:

1. Obtaining the Nonce Value of the Account Initiating the Transaction
2. Building Operations
3. Serializing Transactions
4. Signing Transactions
5. Commiting Transactions

2.4.5.1 Obtaining the Nonce Value of the Account Initiating the Transaction

The developer can maintain the nonce value of each account, and automatically increments by 1 for the nounce value after submitting a transaction, so that multiple transactions can be sent in a short time; otherwise, the nonce value of the account must be added 1 after the execution of the previous transaction is completed. The specific interface call is as follows:

```
// Initialize request parameters
String senderAddress = "buQnnUEBREw2hB6pWHGPzwanX7d28xk6KVcp";
AccountGetNonceRequest getNonceRequest = new AccountGetNonceRequest();
getNonceRequest.setAddress(senderAddress);
```

(continues on next page)

(continued from previous page)

```
// Call the getNonce interface
AccountGetNonceResponse getNonceResponse = sdk.getAccountService().
↪getNonce(getNonceRequest);

// Assign nonce value
if (getNonceResponse.getErrorCode() == 0) {
AccountGetNonceResult result = getNonceResponse.getResult();
System.out.println("nonce: " + result.getNonce());
}
else {
System.out.println("error" + getNonceResponse.getErrorDesc());
}
```

2.4.5.2 Building Operations

The operations refer to some of the actions that are done in the transaction to facilitate serialization of transactions and evaluation of fees. For example, to build an operation to send BU (BUSendOperation), the specific interface call is as follows:

```
String senderAddress = "buQnnUEBREw2hB6pWHGPzwanX7d28xk6KVcp";
String destAddress = "buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw";
Long buAmount = ToBaseUnit.BU2MO("10.9");

BUSendOperation operation = new BUSendOperation();
operation.setSourceAddress(senderAddress);
operation.setDestAddress(destAddress);
operation.setAmount(buAmount);
```

2.4.5.3 Serializing Transactions

The transaction serialization interface is used to serialize transactions and generate transaction blob strings for network transmission. The nonce value and operation are obtained from the interface called, and the specific interface call is as follows:

```
// Initialize variables
String senderAddress = "buQnnUEBREw2hB6pWHGPzwanX7d28xk6KVcp";
Long gasPrice = 1000L;
Long feeLimit = ToBaseUnit.BU2MO("0.01");

// Initialize request parameters
TransactionBuildBlobRequest buildBlobRequest = new TransactionBuildBlobRequest();
buildBlobRequest.setSourceAddress(senderAddress);
buildBlobRequest.setNonce(nonce + 1);
buildBlobRequest.setFeeLimit(feeLimit);
buildBlobRequest.setGasPrice(gasPrice);
buildBlobRequest.addOperation(operation);

// Call the buildBlob interface
TransactionBuildBlobResponse buildBlobResponse = sdk.getTransactionService().
↪buildBlob(buildBlobRequest);
if (buildBlobResponse.getErrorCode() == 0) {
TransactionBuildBlobResult result = buildBlobResponse.getResult();
System.out.println("txHash: " + result.getHash() + ", blob: " + result.
↪getTransactionBlob());
```

(continues on next page)

(continued from previous page)

```
} else {
System.out.println("error: " + buildBlobResponse.getErrorDesc());
}
```

2.4.5.4 Signing Transactions

The `signature transaction` interface is used by the transaction initiator to sign the transaction using the private key of the account. The transactionBlob is obtained from the interface called. The specific interface call is as follows:

```
// Initialize request parameters
String senderPrivateKey = "privbyQCRp7DLqKtRFCqKQJr81TurTqG6UKXMMtGAmPG3abcM9XHjWvq";
String []signerPrivateKeyArr = {senderPrivateKey};
TransactionSignRequest signRequest = new TransactionSignRequest();
signRequest.setBlob(transactionBlob);
for (int i = 0; i < signerPrivateKeyArr.length; i++) {
signRequest.addPrivateKey(signerPrivateKeyArr[i]);
}

// Call the sign interface
TransactionSignResponse signResponse = sdk.getTransactionService().sign(signRequest);
if (signResponse.getErrorCode() == 0) {
TransactionSignResult result = signResponse.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + signResponse.getErrorDesc());
}
```

2.4.5.5 Submitting Transactions

The submit interface is used to send a transaction request to the BU blockchain, triggering the execution of the transaction. transactionBlob and signResult are obtained from the interfaces called. The specific interface call is as follows:

```
// Initialize request parameters
TransactionSubmitRequest submitRequest = new TransactionSubmitRequest();
submitRequest.setTransactionBlob(transactionBlob);
submitRequest.setSignatures(signResult.getSignatures());

// Call the submit interface
TransactionSubmitResponse response = sdk.getTransactionService().
↪submit(submitRequest);
if (0 == response.getErrorCode()) {
System.out.println("Broadcast transactions successfullyhash=" + response.getResult().
↪getHash());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.5 Account Services

Account Services provide account-related interfaces, which include six interfaces: `checkValid`, `getInfo`, `getNonce`, `getBalance`, `getAssets` and `getMetadata`.

2.5.1 checkValid

The `checkValid` interface is used to check the validity of the account address on the blockchain.

The method call is as follows:

```
AccounCheckValidResponse checkValid(AccountCheckValidRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
address	String	Required, the account address to be checked on the blockchain

The response data is shown in the following table:

Parameter	Type	Description
isValid	String	Whether the response data is valid

The error code is shown in the following table:

Exception	Error Code	Description
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
String address = "buQemmMwmRQY1JkcU7w3nhruoX5N3j6C29uo";
AccountCheckValidRequest request = new AccountCheckValidRequest();
request.setAddress(address);

// Call the checkValid interface
AccountCheckValidResponse response = sdk.getAccountService().checkValid(request);
if(0 == response.getErrorCode()) {
System.out.println(response.getResult().isValid());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.5.2 getInfo

The `getInfo` interface is used to obtain the specified account information.

The method call is as follows:

```
AccountGetInfoResponse GetInfo(AccountGetInfoRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
address	String	Required, the account address to be queried on the blockchain

The response data is shown in the following table:

Parameter	Type	Description
address	String	Account address
balance	Long	Account balance, unit is MO, 1 BU = 10^8 MO, the account balance must be > 0
nonce	Long	Account transaction serial number must be greater than 0
priv	*Priv*	Account privilege

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_ADDRESS_ERROR	11006	Invalid address
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters

String accountAddress = "buQemmMwmRQY1JkcU7w3nhruoX5N3j6C29uo";
AccountGetInfoRequest request = new AccountGetInfoRequest();
request.setAddress(accountAddress);

// Call the getInfo interface
AccountGetInfoResponse response = sdk.getAccountService().getInfo(request);
if (response.getErrorCode() == 0) {
AccountGetInfoResult result = response.getResult();
System.out.println("Account info: \n" + JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.5.2.1 Priv

The specific information of Priv is shown in the following table:

Member	Type	Description
masterWeight	Long	Account weight, size limit[0, (Integer.MAX_VALUE * 2L + 1)]
signers	*Signer* []	Signer weight list
threshold	*Threshold*	Threshold

2.5.2.2 Signer

The specific information of Signer is shown in the following table:

Member	Type	Description
address	String	The account address of the signer on the blockchain
weight	Long	Signer weight, size limit[0, (Integer.MAX_VALUE * 2L + 1)]

2.5.2.3 Threshold

The specific information of Signer is shown in the following table:

Member	Type	Description
txThreshold	Long	Transaction default threshold, size limit[0, Long.MAX_VALUE]
typeThresholds	*TypeThreshold*[]	Thresholds for different types of transactions

2.5.2.4 TypeThreshold

The specific information of Signer is shown in the following table:

Member	Type	Description
type	Long	The operation type must be greater than 0
threshold	Long	Threshold, size limit[0, Long.MAX_VALUE]

2.5.3 getNonce

The `getNonce` interface is used to obtain the nonce value of the specified account.

The method call is as follows:

```
AccountGetNonceResponse getNonce(AccountGetNonceRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
address	String	Required, the account address to be queried on the blockchain

The response data is shown in the following table:

Parameter	Type	Description
nonce	Long	Account transaction serial number

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_ADDRESS_ERROR	11006	Invalid address
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters

String accountAddress = "buQswSaKDACkrFsnP1wcVsLAUzXQsemauEjf";
AccountGetNonceRequest request = new AccountGetNonceRequest();
request.setAddress(accountAddress);

```

(continues on next page)

(continued from previous page)

```
// Call the getNonce interface
AccountGetNonceResponse response = sdk.getAccountService().getNonce(request);
if(0 == response.getErrorCode()){
System.out.println("Account nonce:" + response.getResult().getNonce());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.5.4 getBalance

The `getBalance` interface is used to obtain the BU balance of the specified account.

The method call is as follows:

```
AccountGetBalanceResponse getBalance(AccountGetBalanceRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
address	String	Required, the account address to be queried on the blockchain

The response data is shown in the following table:

Parameter	Type	Description
balance	Long	BU balance, unit MO, 1 BU = 10^8 MO

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_ADDRESS_ERROR	11006	Invalid address
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters

String accountAddress = "buQswSaKDACkrFsnP1wcVsLAUzXQsemauEjf";
AccountGetBalanceRequest request = new AccountGetBalanceRequest();
request.setAddress(accountAddress);

// Call the getBalance interface
AccountGetBalanceResponse response = sdk.getAccountService().getBalance(request);
if(0 == response.getErrorCode()){
AccountGetBalanceResult result = response.getResult();
System.out.println("BU balance" + ToBaseUnit.MO2BU(result.getBalance().toString()) +
↪" BU");
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.5.5 getAssets

The `getAssets` interface is used to get all the asset information of the specified account.

The method call is as follows:

```
AccountGetAssets getAssets(AccountGetAssetsRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
address	String	Required, the account address to be queried

The response data is shown in the following table:

Parameter	Type	Description
asset	*AssetInfo*[]	Account asset

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_ADDRESS_ERROR	11006	Invalid address
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
NO_ASSET_ERROR	11009	The account does not have the asset
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
AccountGetAssetsRequest request = new AccountGetAssetsRequest();
request.setAddress("buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw");

// Call the getAssets interface
AccountGetAssetsResponse response = sdk.getAccountService().getAssets(request);
if (response.getErrorCode() == 0) {
AccountGetAssetsResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.5.5.1 AssetInfo

The specific information of AssetInfo is shown in the following table:

Member	Type	Description
key	*Key*	Unique identifier for asset
assetAmount	Long	Amount of assets

2.5.5.2 Key

The specific information of Key is shown in the following table:

Member	Type	Description
code	String	Asset code
issuer	String	The account address for issuing assets

2.5.6 getMetadata

The `getMetadata` interface is used to obtain the metadata information of the specified account.

The method call is as follows:

```
AccountGetMetadataResponse getMetadata(AccountGetMetadataRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
address	String	Required, the account address to be queried
key	String	Optional, metadata keyword, length limit [1, 1024]

The response data is shown in the following table:

Parameter	Type	Description
metadata	*MetadataInfo*	Account

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_ADDRESS_ERROR	11006	Invalid address
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
NO_METADATA_ERROR	11010	The account does not have the metadata
INVALID_DATAKEY_ERROR	11011	The length of key must be between 1 and 1024
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
String accountAddress = "buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw";
AccountGetMetadataRequest request = new AccountGetMetadataRequest();
request.setAddress(accountAddress);
request.setKey("20180704");

// Call the getMetadata interface
AccountGetMetadataResponse response = sdk.getAccountService().getMetadata(request);
if (response.getErrorCode() == 0) {
AccountGetMetadataResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.5.6.1 MetadataInfo

The specific information of MetadataInfo is shown in the following table:

Member	Type	Description
key	String	Metadata keyword
value	String	Metadata content
version	Long	Metadata version

2.6 Asset Services

Asset Services follow the ATP 1.0 protocol, and Account Services provide an asset-related interface. Currently there is one interface: `getInfo`.

2.6.1 getInfo

The `getInfo` interface is used to obtain the specified asset information of the specified account.

The method call is as follows:

```
AssetGetInfoResponse getInfo(AssetGetInfoRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
address	String	Required, the account address to be queried
code	String	Required, asset code, length limit [1, 64]
issuer	String	Required, the account address for issuing assets

The response data is shown in the following table:

Parameter	Type	Description
asset	*AssetInfo*[]	Account asset

The error code is shown in the following table:

The specific example is as follows:

```
// Initialize request parameters

AssetGetInfoRequest request = new AssetGetInfoRequest();
request.setAddress("buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw");
request.setIssuer("buQBjJD1BSJ7nzAbzdTenAhpFjmxRVEEtmxH");
request.setCode("HNC");

// Call the getInfo interface
AssetGetInfoResponse response = sdk.getAssetService().getInfo(request);
if (response.getErrorCode() == 0) {
AssetGetInfoResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
```

(continues on next page)

(continued from previous page)

```
System.out.println("error: " + response.getErrorDesc());
}
```

2.7 Ctp10Token Services

Ctp10Token Services follow the CTP 1.0 protocol and mainly provide contract Token-related interfaces. Currently there are 8 interfaces: `checkValid`, `allowance`, `getInfo`, `getName`, `getSymbol`, `getDecimals`, `getTotalSupply`, and `getBalance`.

2.7.1 checkValid

The `checkValid` interface is used to verify the validity of the contract token.

The method call is as follows:

```
Ctp10TokenCheckValidResponse checkValid(Ctp10TokenCheckValidRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
contractAddress	String	Required, contract address of token to be verified

The response data is shown in the following table:

Parameter	Type	Description
isValid	String	Whether the response data is valid

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
Ctp10TokenCheckValidRequest request = new Ctp10TokenCheckValidRequest();
request.setContractAddress("buQfnVYgXuMo3rvCEpKA6SfRrDpaz8D8A9Ea");

// Call the checkValid interface
Ctp10TokenCheckValidResponse response = sdk.getTokenService().checkValid(request);
if (response.getErrorCode() == 0) {
Ctp10TokenCheckValidResult result = response.getResult();
System.out.println(result.getValid());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.7.2 allowance

The `allowance` interface is used to obtain the amount that the spender allows to extract from the owner.

The method call is as follows:

```
Ctp10TokenAllowanceResponse allowance(Ctp10TokenAllowanceRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
contractAddress	String	Required, contract account address
tokenOwner	String	Required, the account address holding the contract Token
spender	String	Required, authorized account address

The response data is shown in the following table:

Parameter	Type	Description
allowance	String	Allowed amount to be withdrawn

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
NO_SUCH_TOKEN_ERROR	11030	No such token
INVALID_TOKENOWNER_ERRPR	11035	Invalid token owner
INVALID_SPENDER_ERROR	11043	Invalid spender
GET_ALLOWNANCE_ERROR	11036	Failed to get allowance
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
Ctp10TokenAllowanceRequest request = new Ctp10TokenAllowanceRequest();
request.setContractAddress("buQhdBSkJqERBSsYiUShUZFMZQhXvkdNgnYq");
request.setTokenOwner("buQnnUEBREw2hB6pWHGPzwanX7d28xk6KVcp");
request.setSpender("buQnnUEBREw2hB6pWHGPzwanX7d28xk6KVcp");

// Call the allowance interface
Ctp10TokenAllowanceResponse response = sdk.getTokenService().allowance(request);
if (response.getErrorCode() == 0) {
Ctp10TokenAllowanceResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.7.3 getInfo-Ctp10Token

The `getInfo-Ctp10Token` interface is used to obtain information about the contract token.

The method call is as follows:

```
Ctp10TokenGetInfoResponse getInfo(Ctp10TokenGetInfoRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
contractAddress	String	Contract token address to be queried

The response data is shown in the following table:

Parameter	Type	Description
ctp	String	Contract Token version number
symbol	String	Contract Token symbol
decimals	Integer	Accuracy of the number of contracts
totalSupply	String	Total supply of contracts
name	String	The name of the contract Token
contractOwner	String	Owner of the contract Token

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
NO_SUCH_TOKEN_ERROR	11030	No such token
GET_TOKEN_INFO_ERROR	11066	Failed to get token info
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
Ctp10TokenGetInfoRequest request = new Ctp10TokenGetInfoRequest();
request.setContractAddress("buQhdBSkJqERBSsYiUShUZFMZQhXvkdNgnYq");

// Call the allowance interface
Ctp10TokenGetInfoResponse response = sdk.getTokenService().getInfo(request);
if (response.getErrorCode() == 0) {
Ctp10TokenGetInfoResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.7.4 getName

The `getName` interface is used to get the name of the contract Token.

The method call is as follows:

```
Ctp10TokenGetNameResponse getName(Ctp10TokenGetNameRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
contractAddress	String	Contract account address to be queried

The response data is shown in the following table:

Parameter	Type	Description
name	String	The name of the contract Token

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
NO_SUCH_TOKEN_ERROR	11030	No such token
GET_TOKEN_INFO_ERROR	11066	Failed to get token info
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
Ctp10TokenGetNameRequest request = new Ctp10TokenGetNameRequest();
request.setContractAddress("buQhdBSkJqERBSsYiUShUZFMZQhXvkdNgnYq");

// Call the getName interface
Ctp10TokenGetNameResponse response = sdk.getTokenService().getName(request);
if (response.getErrorCode() == 0) {
Ctp10TokenGetNameResult result = response.getResult();
System.out.println(result.getName());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.7.5 getSymbol

The `getSymbol` interface is used to get the symbol of the contract Token.

The method call is as follows:

```
Ctp10TokenGetSymbolResponse getSymbol (Ctp10TokenGetSymbolRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
contractAddress	String	Contract account address to be queried

The response data is shown in the following table:

Parameter	Type	Description
symbol	String	Contract Token symbol

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
NO_SUCH_TOKEN_ERROR	11030	No such token
GET_TOKEN_INFO_ERROR	11066	Failed to get token info
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters

Ctp10TokenGetSymbolRequest request = new Ctp10TokenGetSymbolRequest();
request.setContractAddress("buQhdBSkJqERBSsYiUShUZFMZQhXvkdNgnYq");

// Call the getSymbol interface
Ctp10TokenGetSymbolResponse response = sdk.getTokenService().getSymbol(request);
if (response.getErrorCode() == 0) {
Ctp10TokenGetSymbolResult result = response.getResult();
System.out.println(result.getSymbol());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.7.6 getDecimals

The `getDecimals` interface is used to get the precision of the contract Token.

The method call is as follows:

```
Ctp10TokenGetDecimalsResponse getDecimals (Ctp10TokenGetDecimalsRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
contractAddress	String	Contract account address to be queried

The response data is shown in the following table:

Parameter	Type	Description
decimals	Integer	Contract token precision

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
NO_SUCH_TOKEN_ERROR	11030	No such token
GET_TOKEN_INFO_ERROR	11066	Failed to get token info
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters

Ctp10TokenGetDecimalsRequest request = new Ctp10TokenGetDecimalsRequest();
request.setContractAddress("buQhdBSkJqERBSsYiUShUZFMZQhXvkdNgnYq");

// Call the getDecimals interface
Ctp10TokenGetDecimalsResponse response = sdk.getTokenService().getDecimals(request);
if (response.getErrorCode() == 0) {
Ctp10TokenGetDecimalsResult result = response.getResult();
System.out.println(result.getDecimals());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.7.7 getTotalSupply

The `getTotalSupply` interface is used to get the total supply of contract tokens.

The method call is as follows:

```
Ctp10TokenGetTotalSupplyResponse getTotalSupply(Ctp10TokenGetTotalSupplyRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
contractAddress	String	Contract account address to be queried

The response data is shown in the following table:

Parameter	Type	Description
totalSupply	String	Total supply of contract Token

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
NO_SUCH_TOKEN_ERROR	11030	No such token
GET_TOKEN_INFO_ERROR	11066	Failed to get token info
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
Ctp10TokenGetTotalSupplyRequest request = new Ctp10TokenGetTotalSupplyRequest();
request.setContractAddress("buQhdBSkJqERBSsYiUShUZFMZQhXvkdNgnYq");

// Call the getDecimals interface
Ctp10TokenGetTotalSupplyResponse response = sdk.getTokenService().
↪getTotalSupply(request);
if (response.getErrorCode() == 0) {
Ctp10TokenGetTotalSupplyResult result = response.getResult();
```

(continues on next page)

(continued from previous page)

```
System.out.println(result.getTotalSupply());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.7.8 getBalance-Ctp10Token

The `getBalance-Ctp10Token` interface is used to get the account balance of the contract Token owner.

The method call is as follows:

```
Ctp10TokenGetBalanceResponse getBalance(Ctp10TokenGetBalanceRequest)
```

The request parameters are shown in the following table:

Parameter	Type	Description
contractAddress	String	Contract account address to be queried
tokenOwner	String	Required, the account address holding the contract Token

The response data is shown in the following table:

Parameter	Type	Description
balance	Long	Token balance

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_TOKENOWNER_ERRPR	11035	Invalid token owner
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
NO_SUCH_TOKEN_ERROR	11030	No such token
GET_TOKEN_INFO_ERROR	11066	Failed to get token info
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
Ctp10TokenGetBalanceRequest request = new Ctp10TokenGetBalanceRequest();
request.setContractAddress("buQhdBSkJqERBSsYiUShUZFMZQhXvkdNgnYq");
request.setTokenOwner("buQnnUEBREw2hB6pWHGPzwanX7d28xk6KVcp");

// Call the getBalance interface
Ctp10TokenGetBalanceResponse response = sdk.getTokenService().getBalance(request);
if (response.getErrorCode() == 0) {
Ctp10TokenGetBalanceResult result = response.getResult();
System.out.println(result.getBalance());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.8 Contract Services

Contract Services provide contract-related interfaces and currently have four interfaces: `checkValid`, `getInfo`, `getAddress`, and `call`.

2.8.1 checkValid

The `checkValid` interface is used to check the validity of the contract account.

The method call is as follows:

```
ContractCheckValidResponse checkValid(ContractCheckValidRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
contractAddress	String	Contract account address to be tested

The response data is shown in the following table:

Parameter	Type	Description
isValid	Boolean	Whether the response data is valid

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS_ERROR	11037	Invalid contract address
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
ContractCheckValidRequest request = new ContractCheckValidRequest();
request.setContractAddress("buQfnVYgXuMo3rvCEpKA6SfRrDpaz8D8A9Ea");

// Call the getDecimals interface
ContractCheckValidResponse response = sdk.getContractService().checkValid(request);
if (response.getErrorCode() == 0) {
ContractCheckValidResult result = response.getResult();
System.out.println(result.getValid());
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.8.2 getInfo

The `getInfo` interface is used to query the contract code.

The method call is as follows:

```
ContractGetInfoResponse getInfo (ContractGetInfoRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
contractAddress	String	Contract account address to be queried

The response data is shown in the following table:

Parameter	Type	Description
contract	ContractInfo	Contract info

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_CONTRACTADDRESS _ERROR	11037	Invalid contract address
CONTRACTADDRESS_NOT_CON TRACTAC-COUNT_ERROR	11038	contractAddress is not a contract account
NO_SUCH_TOKEN_ERROR	11030	No such token
GET_TOKEN_INFO_ERROR	11066	Failed to get token info
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
ContractGetInfoRequest request = new ContractGetInfoRequest();
request.setContractAddress("buQfnVYgXuMo3rvCEpKA6SfRrDpaz8D8A9Ea");

// Call the getInfo interface
ContractGetInfoResponse response = sdk.getContractService().getInfo(request);
if (response.getErrorCode() == 0) {
System.out.println(JSON.toJSONString(response.getResult(), true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.8.2.1 ContractInfo

The specific information of ContractInfo is shown in the following table:

Member	Type	Description
type	Integer	Contract type, default is 0
payload	String	Contract code

2.8.3 getAddress

The `getAddress` interface is used to query the contract address.

The method call is as follows:

```
ContractGetAddressResponse getInfo (ContractGetAddressRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
hash	String	The hash used to create a contract transaction

The response data is shown in the following table:

Parameter	Type	Description
contractAddressList	List (ContractAddressInfo)	Contract address list

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_HASH_ERROR	11055	Invalid transaction hash
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
ContractGetAddressRequest request = new ContractGetAddressRequest();
request.setHash("44246c5ba1b8b835a5cbc29bdc9454cdb9a9d049870e41227f2dcfbcf7a07689");

// Call the getAddress interface
ContractGetAddressResponse response = sdk.getContractService().getAddress(request);
if (response.getErrorCode() == 0) {
System.out.println(JSON.toJSONString(response.getResult(), true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.8.3.1 ContractAddressInfo

The specific information of ContractAddressInfo is shown in the following table:

Member	Type	Description
contractAddress	String	Contract address
operationIndex	Integer	The subscript of the operation

2.8.4 call

The call `interface` is used to debug the contract code.

The method call is as follows:

```
ContractCallesponse call(ContractCallRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
sourceAddress	String	Optional, the account address to trigger the contract
contractAddress	String	Optional, the contract account address and code cannot be empty at the same time
code	String	Optional, the contract code and contractAddress cannot be empty at the same time, length limit [1, 64]
input	String	Optional, input parameter for the contract
contractBalance	String	Optional, the initial BU balance given to the contract, unit MO, 1 BU = 10^8 MO, size limit [1, Long.MAX_VALUE]
optType	Integer	Required, 0: Call the read/write interface of the contract init, 1: Call the read/write interface of the contract main, 2: Call the read-only interface query
feeLimit	Long	Minimum fee required for the transaction, size limit [1, Long.MAX_VALUE]
gasPrice	Long	Transaction fuel price, size limit [1000, Long.MAX_VALUE]

The response data is shown in the following table:

Parameter	Type	Description
logs	JSONObject	Log information
queryRets	JSONArray	Query the result set
stat	*ContractStat*	Contract resource occupancy
txs	*TransactionEnvs*[]	Transaction set

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_SOURCEADDRESS_E RROR	11002	Invalid sourceAddress
INVALID_CONTRACTADDRESS _ERROR	11037	Invalid contract address
CONTRACTADDRESS_CODE_BO TH_NULL_ERROR	11063	ContractAddress and code cannot be empty at the same time
INVALID_OPTTYPE_ERROR	11064	OptType must be between 0 and 2
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
ContractCallRequest request = new ContractCallRequest();
request.setCode("\"use strict\";log(undefined);function query() {␣
↪getBalance(thisAddress); }");
request.setFeeLimit(1000000000L);
request.setOptType(2);

// Call the ``call`` interface
ContractCallResponse response = sdk.getContractService().call(request);
if (response.getErrorCode() == 0) {
```

(continues on next page)

(continued from previous page)

```
ContractCallResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.8.4.1 ContractStat

The specific information of ContractStat is shown in the following table:

Member	Type	Description
applyTime	Long	Receipt time
memoryUsage	Long	Memory footprint
stackUsage	Long	Stack occupancy
step	Long	Steps needed

2.8.4.2 TransactionEnvs

The specific information of TransactionEnvs is shown in the following table:

Member	Type	Description
transactionEnv	*TransactionEnv*	Transaction

2.8.4.3 TransactionEnv

The specific information of TransactionEnv is shown in the following table:

Member	Type	Description
transaction	*TransactionInfo*	Transaction content
trigger	*ContractTrigger*	Contract trigger

2.8.4.4 TransactionInfo

The specific information of TransactionInfo is shown in the following table:

Member	Type	Description
sourceAddress	String	The source account address initiating the transaction
feeLimit	Long	Minimum fees required for the transaction
gasPrice	Long	Transaction fuel price
nonce	Long	Transaction serial number
operations	Operation[]	Operation list

2.8.4.5 ContractTrigger

The specific information of ContractTrigger is shown in the following table:

Member	Type	Description
transaction	*TriggerTransaction*	Trigger transactions

2.8.4.6 Operation

The specific information of Operation is shown in the following table:

Member	Type	Description
type	Integer	Operation type
sourceAddress	String	The source account address initiating operations
metadata	String	Note
createAccount	*OperationCreateAccount*	Operation of creating accounts
issueAsset	*OperationIssueAsset*	Operation of issuing assets
payAsset	*OperationPayAsset*	Operation of transferring assets
payCoin	*OperationPayCoin*	Operation of sending BU
setMetadata	*OperationSetMetadata*	Operation of setting metadata
setPrivilege	*OperationSetPrivilege*	Operation of setting account privilege
log	*OperationLog*	Record logs

2.8.4.7 TriggerTransaction

The specific information of TriggerTransaction is shown in the following table:

Member	Type	Description
hash	String	Transaction hash

2.8.4.8 OperationCreateAccount

The specific information of OperationCreateAccount is shown in the following table:

Member	Type	Description
destAddress	String	Target account address
contract	Contract	Contract info
priv	*Priv*	Account privilege
metadata	*MetadataInfo*[]	Account
initBalance	Long	Account assets, unit MO, 1 BU = 10^8 MO,
initInput	String	The input parameter for the init function of the contract

2.8.4.9 Contract

The specific information of Contract is shown in the following table:

Member	Type	Description
type	Integer	The contract language is not assigned value by default
payload	String	The contract code for the corresponding language

2.8.4.10 MetadataInfo

The specific information of MetadataInfo is shown in the following table:

Member	Type	Description
key	String	metadata keyword
value	String	metadata content
version	Long	metadata version

2.8.4.11 OperationIssueAsset

The specific information of OperationIssueAsset is shown in the following table:

Member	Type	Description
code	String	Assets encoding
assetAmount	Long	Assets amount

2.8.4.12 OperationPayAsset

The specific information of OperationPayAsset is shown in the following table:

Member	Type	Description
destAddress	String	The target account address to which the asset is transferred
asset	*AssetInfo*	Account asset
input	String	Input parameters for the main function of the contract

2.8.4.13 OperationPayCoin

The specific information of OperationPayCoin is shown in the following table:

Member	Type	Description
destAddress	String	The target account address to which the asset is transferred
buAmount	Long	BU amounts to be transferred
input	String	Input parameters for the main function of the contract

2.8.4.14 OperationSetMetadata

The specific information of OperationSetMetadata is shown in the following table:

Member	Type	Description
key	String	metadata keyword
value	String	metadata content
version	Long	metadata version
deleteFlag	boolean	Whether to delete metadata

2.8.4.15 OperationSetPrivilege

The specific information of OperationSetPrivilege is shown in the following table:

2.8.4.16 OperationLog

The specific information of OperationLog is shown in the following table:

Member	Type	Description
topic	String	Log theme
data	String[]	Log content

2.9 Transaction Services

Transaction Services provide transaction-related interfaces and currently have five interfaces: `buildBlob`, `evaluateFee`, `sign`, `submit`, and `getInfo`.

2.9.1 buildBlob

The `buildBlob` interface is used to serialize transactions and generate transaction blob strings for network transmission.

Before you can call buildBlob, you need to build some operations. There are 16 operations: `AccountActivateOperation`, `AccountSetMetadataOperation`, `AccountSetPrivilegeOperation`, `AssetIssueOperation`, `AssetSendOperation`, `BUSendOperation`, `TokenIssueOperation`, `TokenTransferOperation`, `TokenTransferFromOperation`, `TokenApproveOperation`, `TokenAssignOperation`, `TokenChangeOwnerOperation`, `ContractCreateOperation`, `ContractInvokeByAssetOperation`, `ContractInvokeByBUOperation`, and `LogCreateOperation`.

The method call is as follows:

```
TransactionBuildBlobResponse buildBlob(TransactionBuildBlobRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
sourceAddress	String	Required, the source account address initiating the operation
nonce	Long	Required, the transaction serial number to be initiated, add 1 in the function, size limit [1, Long.MAX_VALUE]
gasPrice	Long	Required, transaction gas price, unit MO, 1 BU = 10^8 MO, size limit [1000, Long.MAX_VALUE]
feeLimit	Long	Required, the minimum fees required for the transaction, unit MO, 1 BU = 10^8 MO, size limit [1, Long.MAX_VALUE]
operation	BaseOperation[]	Required, list of operations to be committed which cannot be empty
ceilLedgerSeq	Long	Optional, set a value which will be combined with the current block height to restrict transactions. If transactions do not complete within the set value plus the current block height, the transactions fail. The value you set must be greater than 0. If the value is set to 0, no limit is set.
metadata	String	Optional, note

The response data is shown in the following table:

Parameter	Type	Description
transactionBlob	String	Serialized transaction hex string
hash	String	Transaction hash

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_SOURCEADDRESS_E RROR	11002	Invalid sourceAddress
INVALID_NONCE_ERROR	11048	Nonce must be between 1 and Long.MAX_VAL
INVALID_DESTADDRESS_ERR OR	11003	Invalid destAddress
INVALID_INITBALANCE_ERR OR	11004	InitBalance must be between 1 and Long.MAX_
SOURCEADDRESS_EQUAL_DES TADDRESS_ERROR	11005	SourceAddress cannot be equal to destAddress
INVALID_ISSUE_AMMOUNT_E RROR	11008	AssetAmount that will be issued must be betwee
INVALID_DATAKEY_ERROR	11011	The length of key must be between 1 and 1024
INVALID_DATAVALUE_ERROR	11012	The length of value must be between 0 and 2560
INVALID_DATAVERSION_ERR OR	11013	The version must be equal to or greater than 0
INVALID_MASTERWEIGHT _ERROR	11015	MasterWeight must be between 0 and (Integer.M
INVALID_SIGNER_ADDRESS _ERROR	11016	Invalid signer address
INVALID_SIGNER_WEIGHT _ERROR	11017	Signer weight must be between 0 and (Integer.M
INVALID_TX_THRESHOLD_ER ROR	11018	TxThreshold must be between 0 and Long.MAX
INVALID_OPERATION_TYPE _ERROR	11019	Operation type must be between 1 and 100
INVALID_TYPE_THRESHOLD _ERROR	11020	TypeThreshold must be between 0 and Long.MA
INVALID_ASSET_CODE _ERROR	11023	The length of key must be between 1 and 64
INVALID_ASSET_AMOUNT_ER ROR	11024	AssetAmount must be between 0 and Long.MA
INVALID_BU_AMOUNT_ERROR	11026	BuAmount must be between 0 and Long.MAX_
INVALID_ISSUER_ADDRESS _ERROR	11027	Invalid issuer address
NO_SUCH_TOKEN_ERROR	11030	No such token
INVALID_TOKEN_NAME_ERRO R	11031	The length of token name must be between 1 an
INVALID_TOKEN_SYMBOL_ER ROR	11032	The length of symbol must be between 1 and 10
INVALID_TOKEN_DECIMALS _ERROR	11033	Decimals must be between 0 and 8
INVALID_TOKEN_TOTALSUPP LY_ERROR	11034	TotalSupply must be between 1 and Long.MAX_
INVALID_TOKENOWNER_ERRP R	11035	Invalid token owner
INVALID_CONTRACTADDRESS _ERROR	11037	Invalid contract address
CONTRACTADDRESS_NOT_CON TRACTACCOUNT_ERROR	11038	ContractAddress is not a contract account
INVALID_TOKEN_AMOUNT_ER ROR	11039	Token amount must be between 1 and Long.MA
SOURCEADDRESS_EQUAL_CON TRACTADDRESS_ERROR	11040	SourceAddress cannot be equal to contractAddr
INVALID_FROMADDRESS_ERR OR	11041	Invalid fromAddress
FROMADDRESS_EQUAL_DESTA DDRESS_ERROR	11042	FromAddress cannot be equal to destAddress
INVALID_SPENDER_ERROR	11043	Invalid spender
PAYLOAD_EMPTY_ERROR	11044	Payload cannot be empty
INVALID_LOG_TOPIC _ERROR	11045	The length of key must be between 1 and 128
INVALID_LOG_DATA _ERROR	11046	The length of value must be between 1 and 1024
INVALID_CONTRACT_TYPE_E RROR	11047	Type must be equal to or greater than 0
INVALID_NONCE_ERROR	11048	Nonce must be between 1 and Long.MAX_VAL
INVALID_ GASPRICE_ERROR	11049	GasPrice must be between 1000 and Long.MAX
INVALID_FEELIMIT_ERROR	11050	FeeLimit must be between 1 and Long.MAX_V
OPERATIONS_EMPTY_ERROR	11051	Operations cannot be empty
INVALID_CEILLEDGERSEQ_E RROR	11052	CeilLedgerSeq must be equal or greater than 0
OPERATIONS_ONE_ERROR	11053	One of the operations cannot be resolved

Table 1 – continued from previous page

Exception	Error Code	Description
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize variables
String senderAddresss = "buQfnVYgXuMo3rvCEpKA6SfRrDpaz8D8A9Ea";
String destAddress = "buQsurH1M4rjLkfjzkxR9KXJ6jSu2r9xBNEw";
Long buAmount = ToBaseUnit.BU2MO("10.9");
Long gasPrice = 1000L;
Long feeLimit = ToBaseUnit.BU2MO("0.01");
Long nonce = 1L;

// Build the sendBU operation
BUSendOperation operation = new BUSendOperation();
operation.setSourceAddress(senderAddresss);
operation.setDestAddress(destAddress);
operation.setAmount(buAmount);

// Initialize request parameters
TransactionBuildBlobRequest request = new TransactionBuildBlobRequest();
request.setSourceAddress(senderAddresss);
request.setNonce(nonce);
request.setFeeLimit(feeLimit);
request.setGasPrice(gasPrice);
request.addOperation(operation);

// Call the buildBlob interface
String transactionBlob = null;
TransactionBuildBlobResponse response = sdk.getTransactionService().
↪buildBlob(request);
if (response.getErrorCode() == 0) {
TransactionBuildBlobResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.9.1.1 BaseOperation

BaseOperation is the base class for all operations in the `buildBlob` interface. The following table describes BaseOperation:

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
metadata	String	Optional, note

2.9.1.2 AccountActivateOperation

AccountActivateOperation inherits from BaseOperation, and feeLimit is currently fixed at 0.01 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
destAddress	String	Required, target account address
initBalance	Long	Required, initialize the asset, unit MO, 1 BU = 10^8 MO, size (0, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.3 AccountSetMetadataOperation

AccountSetMetadataOperation is inherited from BaseOperation, and feeLimit is currently fixed at 0.01 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
key	String	Required, metadata keyword, length limit [1, 1024]
value	String	Required, metadata content, length limit [0, 256000]
version	Long	Optional, metadata version
deleteFlag	Boolean	Optional, whether to delete metadata
metadata	String	Optional, note

2.9.1.4 AccountSetPrivilegeOperation

AccountSetPrivilegeOperation inherits from BaseOperation, and feeLimit is currently fixed at 0.01 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
masterWeight	String	Optional, account weight, size limit [0, (Integer.MAX_VALUE * 2L + 1)]
signers	Signer[]	Optional, signer weight list
txThreshold	String	Optional, transaction threshold, size limit [0, Long.MAX_VALUE]
typeThreshold	TypeThreshold[]	Optional, specify transaction threshold
metadata	String	Optional, note

2.9.1.5 AssetIssueOperation

AssetIssueOperation inherits from BaseOperation, and feeLimit is currently fixed at 50.01 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
code	String	Required, asset code, length limit [1, 64]
assetAmount	Long	Required, number of asset issues, size limit [0, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.6 AssetSendOperation

AssetSendOperation inherits from BaseOperation, and feeLimit is currently fixed at 0.01 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
destAddress	String	Required, target account address
code	String	Required, asset code, length limit [1, 64]
issuer	String	Required, account address issuing assets
assetAmount	Long	Required, asset quantity, size limit [0, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.7 BUSendOperation

BUSendOperation inherits from BaseOperation, feeLimit is currently (2018.07.26) fixed at 0.01 BU.

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
destAddress	String	Required, target account address
buAmount	Long	Required, amount of asset issued, size limit [0, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.8 Ctp10TokenIssueOperation

Ctp10TokenIssueOperation inherits from BaseOperation, and feeLimit is currently fixed at 10.08 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
initBalance	Long	Required, initial assets for the contract account, unit MO, 1 BU = 10^8 MO, size limit [1, max(64)]
name	String	Required, ctp10Token name, length limit [1, 1024]
symbol	String	Required, ctp10Token symbol, length limit [1, 1024]
decimals	Integer	Required, the precision of the number of ctp10Token, size limit [0, 8]
supply	String	Required, total supply issued by ctp10Token (without precision), size limit [1, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.9 Ctp10TokenTransferOperation

Ctp10TokenTransferOperation inherits from BaseOperation, and feeLimit currently (2018.07.26) is fixed at 0.02 BU.

Member	Type	Description
sourceAddress	String	Optional, account address holding the contract token
contractAddress	String	Required, contract account address
destAddress	String	Required, target account address to which token is transferred
tokenAmount	String	Required, amount of tokens to be transferred, size limit [1, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.10 TokenTransferFromOperation

TokenTransferFromOperation inherits from BaseOperation, and feeLimit is currently fixed at 0.02 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
contractAddress	String	Required, contract account address
fromAddress	String	Required, source account address from which token is transferred
destAddress	String	Required, target account address to which token is transferred
tokenAmount	String	Required, amount of ctp10Tokens to be transferred, size limit [1, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.11 Ctp10TokenApproveOperation

Ctp10TokenApproveOperation inherits from BaseOperation, and feeLimit is currently fixed at 0.02 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, account address holding the contract token
contractAddress	String	Required, contract account address
spender	String	Required, authorized account address
tokenAmount	String	Required, the number of authorized ctp10Tokens to be transferred, size limit [1, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.12 Ctp10TokenAssignOperation

Ctp10TokenAssignOperation inherits from BaseOperation, feeLimit is currently (2018.07.26) fixed at 0.02 BU.

Member	Type	Description
sourceAddress	String	Optional, account address holding the contract token
contractAddress	String	Required, contract account address
destAddress	String	Required, target account address to be assigned
tokenAmount	String	Required, amount of ctp10Tokens to be allocated, size limit [1, Long.MAX_VALUE]
metadata	String	Optional, note

2.9.1.13 Ctp10TokenChangeOwnerOperation

Ctp10TokenChangeOwnerOperation inherits from BaseOperation, and feeLimit is currently fixed at 0.02 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, account address holding the contract token
contractAddress	String	Required, contract account address
tokenOwner	String	Required, target account address to which token is transferred
metadata	String	Optional, note

2.9.1.14 ContractCreateOperation

ContractCreateOperation inherits from BaseOperation, and feeLimit is currently fixed at 10.01 BU (2018.07.26).

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
initBalance	Long	Required, initial asset for contract account, unit MO, 1 BU = 10^8 MO, size limit [1, Long.MAX_VALUE]
type	Integer	Optional, the language of the contract, the default is 0
payload	String	Required, contract code for the corresponding language
initInput	String	Optional, the input parameters of the init method in the contract code
metadata	String	Optional, note

2.9.1.15 ContractInvokeByAssetOperation

ContractInvokeByAssetOperation inherits from BaseOperation. FeeLimit requires to add fees according to the execution of the transaction in the contract. First, the transaction fee is initiated. At present the fee (2018.07.26) is 0.01BU, and then the transaction in the contract also requires the transaction initiator to add the transaction fees.

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
contractAddress	String	Required, contract account address
code	String	Optional, asset code, length limit [0, 1024]; when it is empty, only the contract is triggered
issuer	String	Optional, the account address issuing assets; when it is null, only trigger the contract
assetAmount	Long	Optional, asset quantity, size limit [0, Long.MAX_VALUE], when it is 0, only trigger the contract
input	String	Optional, the input parameter of the main() method for the contract to be triggered
metadata	String	Optional, note

2.9.1.16 ContractInvokeByBUOperation

ContractInvokeByBUOperation inherits from BaseOperation. FeeLimit requires to add fees according to the execution of the transaction in the contract. First, the transaction fee is initiated. At present the fee (2018.07.26) is 0.01BU, and then the transaction in the contract also requires the transaction initiator to add the transaction fees.

Member	Type	Description
sourceAddress	String	Optional, source account address of the operation
contractAddress	String	Required, contract account address
buAmount	Long	Optional, number of asset issues, size limit [0, Long.MAX_VALUE], when it is 0 only triggers the contract
input	String	Optional, the input parameter of the main() method for the contract to be triggered
metadata	String	Optional, note

2.9.2 evaluateFee

The evaluateFee `interface` implements the cost estimate for the transaction.

The method call is as follows:

```
TransactionEvaluateFeeResponse evaluateFee (TransactionEvaluateFeeRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
sourceAddress	String	Required, the source account address issuing the operation
nonce	Long	Required, transaction serial number to be initiated, size limit [1, Long.MAX_VALUE]
operation	BaseOperation[]	Required, list of operations to be committed which cannot be empty
signtureNumber	Integer	Optional, the number of people to sign, the default is 1, size limit [1, Integer.MAX_VALUE]
ceilLedgerSeq	Long	Optional, set a value which will be combined with the current block height to restrict transactions. If transactions do not complete within the set value plus the current block height, the transactions fail. The value you set must be greater than 0. If the value is set to 0, no limit is set.
metadata	String	Optional, note

The response data is shown in the following table:

Parameter	Type	Description
txs	*TestTx*[]	Evaluation transaction set

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_SOURCEADDRESS_ERROR	11002	Invalid sourceAddress
INVALID_NONCE_ERROR	11045	Nonce must be between 1 and Long.MAX_VALUE
OPERATIONS_EMPTY_ERROR	11051	Operations cannot be empty
OPERATIONS_ONE_ERROR	11053	One of operations cannot be resolved
INVALID_SIGNATURENUMBER_ERROR	11054	SignagureNumber must be between 1 and Integer.MAX_VALU E
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize variables
String senderAddresss = "buQnnUEBREw2hB6pWHGPzwanX7d28xk6KVcp";
```

(continues on next page)

(continued from previous page)

```
String destAddress = "buQfnVYgXuMo3rvCEpKA6SfRrDpaz8D8A9Ea";
Long buAmount = ToBaseUnit.BU2MO("10.9");
Long gasPrice = 1000L;
Long feeLimit = ToBaseUnit.BU2MO("0.01");
Long nonce = 51L;

// Build the sendBU operation
BUSendOperation buSendOperation = new BUSendOperation();
buSendOperation.setSourceAddress(senderAddresss);
buSendOperation.setDestAddress(destAddress);
buSendOperation.setAmount(buAmount);

// Initialize request parameters for transaction evaluation

TransactionEvaluateFeeRequest request = new TransactionEvaluateFeeRequest();
request.addOperation(buSendOperation);
request.setSourceAddress(senderAddresss);
request.setNonce(nonce);
request.setSignatureNumber(1);
request.setMetadata(HexFormat.byteToHex("evaluate fees".getBytes()));

// Call the evaluateFee interface
TransactionEvaluateFeeResponse response = sdk.getTransactionService().
↪evaluateFee(request);
if (response.getErrorCode() == 0) {
TransactionEvaluateFeeResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.9.2.1 TestTx

The specific information of TestTx is shown in the following table:

Member	Type	Description
transactionEnv	*TestTransactionFees*	Assess transaction costs

2.9.2.2 TestTransactionFees

The specific information of TestTransactionFees is shown in the following table:

Member	Type	Description
transactionFees	*TransactionFees*	Transaction fees

2.9.2.3 TransactionFees

The specific information of TransactionFees is shown in the following table:

Member	Type	Description
feeLimit	Long	Minimum fees required for the transaction
gasPrice	Long	Transaction gas price

2.9.3 sign

The `sign` interface is used to implement the signature of the transaction.

The method call is as follows:

```
TransactionSignResponse sign(TransactionSignRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
blob	String	Required, pending transaction blob to be signed
privateKeys	String[]	Required, private key list

The response data is shown in the following table:

Parameter	Type	Description
signatures	Signature	Signed data list

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_BLOB_ERROR	11056	Invalid blob
PRIVATEKEY_NULL_ERROR	11057	PrivateKeys cannot be empty
PRIVATEKEY_ONE_ERROR	11058	One of the privateKeys is invalid
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
String issuePrivateKey = "privbyQCRp7DLqKtRFCqKQJr81TurTqG6UKXMMtGAmPG3abcM9XHjWvq";
String []signerPrivateKeyArr = {issuePrivateKey};
String transactionBlob =
↪"0A246275516E6E5545425245773268423670574847507A77616E5837643238786B364B566370102118C0843D20E8073A5
↪";
TransactionSignRequest request = new TransactionSignRequest();
request.setBlob(transactionBlob);
for (int i = 0; i < signerPrivateKeyArr.length; i++) {
request.addPrivateKey(signerPrivateKeyArr[i]);
}
TransactionSignResponse response = sdk.getTransactionService().sign(request);
if(0 == response.getErrorCode()){
System.out.println(JSON.toJSONString(response.getResult(), true));
}else{
System.out.println("error: " + response.getErrorDesc());
}
```

2.9.3.1 Signature

The specific information of signature is shown in the following table:

Member	Type	Description
signData	Long	Data signed
publicKey	Long	Public key

2.9.4 submit

The `submit` interface is used to implement the submission of the transaction.

The method call is as follows:

```
TransactionSubmitResponse submit(TransactionSubmitRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
blob	String	Required, transaction blob
signature	Signature[]	Required, signature list

The response data is shown in the following table:

Parameter	Type	Description
hash	String	Transaction hash

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_BLOB_ERROR	11056	Invalid blob
SIGNATURE_EMPTY_ERROR	11067	The signatures cannot be empty
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
String transactionBlob =
↪"0A246275516E6E5545425245773268423670574847507A77616E5837643238786B364B566370102118C0843D20E8073A56
↪";
Signature signature = new Signature();
signature.setSignData(
↪"D2B5E3045F2C1B7D363D4F58C1858C30ABBBB0F41E4B2E18AF680553CA9C3689078E215C097086E47A4393BCA715C7A5D2
↪");
signature.setPublicKey(
↪"b0011765082a9352e04678ef38d38046dc01306edef676547456c0c23e270aaed7ffe9e31477");
TransactionSubmitRequest request = new TransactionSubmitRequest();
request.setTransactionBlob(transactionBlob);
request.addSignature(signature);

// Call the submit interface
```

(continues on next page)

(continued from previous page)

```
TransactionSubmitResponse response = sdk.getTransactionService().submit(request);
if (0 == response.getErrorCode()) { // Committed transactions successfully
System.out.println(JSON.toJSONString(response.getResult(), true));
} else{
System.out.println("error: " + response.getErrorDesc());
}
```

2.9.5 getInfo

The `getInfo` interface is used to implement query transactions based on transaction hashes.

The method call is as follows:

```
TransactionGetInfoResponse getInfo (TransactionGetInfoRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
hash	String	Transaction hash

The response data is shown in the following table:

Parameter	Type	Description
totalCount	Long	Total number of transactions returned
transactions	*TransactionHistory* []	Transaction content

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_HASH_ERROR	11055	Invalid transaction hash
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
String txHash = "1653f54fbba1134f7e35acee49592a7c29384da10f2f629c9a214f6e54747705";
TransactionGetInfoRequest request = new TransactionGetInfoRequest();
request.setHash(txHash);

// Call the getInfo interface
TransactionGetInfoResponse response = sdk.getTransactionService().getInfo(request);
if (response.getErrorCode() == 0) {
System.out.println(JSON.toJSONString(response.getResult(), true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.9.5.1 TransactionHistory

The specific information of TransactionHistory is shown in the following table:

Member	Type	Description
actualFee	String	Actual transaction cost
closeTime	Long	Transaction closure time
errorCode	Long	Transaction error code
errorDesc	String	Transaction description
hash	String	Transaction hash
ledgerSeq	Long	Block serial number
transaction	*TransactionInfo*	List of transaction contents
signatures	*Signature*[]	Signature list
txSize	Long	Transaction size

2.10 Block Services

Block services provide block-related interfaces. There are currently 11 interfaces: `getNumber`, `checkStatus`, `getTransactions`, `getInfo`, `getLatestInfo`, `getValidators`, `getLatestValidators`, `getReward`, `getLatestReward`, `getFees` and `getLatestFees`.

2.10.1 getNumber

The `getNumber` interface is used to query the latest block height.

The method call is as follows:

```
BlockGetNumberResponse getNumber();
```

The response data is shown in the following table:

Parameter	Type	Description
header	BlockHeader	Block head
blockNumber	Long	The latest block height,corresponding to the underlying field sequence

The error code is shown in the following table:

Exception	Error Code	Description
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Call the getNumber interface
BlockGetNumberResponse response = sdk.getBlockService().getNumber();
if(0 == response.getErrorCode()){
System.out.println(JSON.toJSONString(response.getResult(), true));
}else{
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.2 checkStatus

The `checkStatus` interface is used to check if the local node block is synchronized.

The method call is as follows:

```
BlockCheckStatusResponse checkStatus();
```

The response data is shown in the following table:

Parameter	Type	Description
isSynchronous	Boolean	Whether the block is synchronized

The error code is shown in the following table:

Exception	Error Code	Description
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Call the checkStatus interface
BlockCheckStatusResponse response = sdk.getBlockService().checkStatus();
if(0 == response.getErrorCode()){
System.out.println(JSON.toJSONString(response.getResult(), true));
}else{
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.3 getTransactions

The `getTransactions` interface is used to query all transactions at the specified block height.

The method call is as follows:

```
BlockGetTransactionsResponse getTransactions(BlockGetTransactionsRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
blockNumber	Long	Required, the height of the block to be queried must be greater than 0

The response data is shown in the following table:

Parameter	Type	Description
totalCount	Long	Total number of transactions returned
transactions	TransactionHistory[]	Transaction content

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_BLOCKNUMBER_ERROR	11060	BlockNumber must be greater than 0
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
Long blockNumber = 617247L;// Block 617247
BlockGetTransactionsRequest request = new BlockGetTransactionsRequest();
request.setBlockNumber(blockNumber);

// Call the getTransactions interface
BlockGetTransactionsResponse response = sdk.getBlockService().
↪getTransactions(request);
if(0 == response.getErrorCode()){
System.out.println(JSON.toJSONString(response.getResult(), true));
}else{
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.4 getInfo

The `getInfo` interface is used to obtain block information.

The method call is as follows:

```
BlockGetInfoResponse getInfo(BlockGetInfoRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
blockNumber	Long	Required, the height of the block to be queried

The response data is shown in the following table:

Parameter	Type	Description
closeTime	Long	Block closure time
number	Long	Block height
txCount	Long	Total transactions amount
version	String	Block version

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_BLOCKNUMBER_ERROR	11060	BlockNumber must be greater than 0
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
BlockGetInfoRequest request = new BlockGetInfoRequest();
request.setBlockNumber(629743L);

// Call the getInfo interface
BlockGetInfoResponse response = sdk.getBlockService().getInfo(request);
if (response.getErrorCode() == 0) {
BlockGetInfoResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.5 getLatestInfo

The `getLatestInfo` interface is used to get the latest block information.

The method call is as follows:

```
BlockGetLatestInfoResponse getLatestInfo();
```

The response data is shown in the following table:

Parameter	Type	Description
closeTime	Long	Block closure time
number	Long	Block height,corresponding to the underlying field seq
txCount	Long	Total transactions amount
version	String	Block version

The error code is shown in the following table:

Exception	Error Code	Description
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Call the getLatestInfo interface
BlockGetLatestInfoResponse response = sdk.getBlockService().getLatestInfo();
if (response.getErrorCode() == 0) {
BlockGetLatestInfoResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.6 getValidators

The `getValidators` interface is used to get the number of all the authentication nodes in the specified block.

The method call is as follows:

```
BlockGetValidatorsResponse getValidators(BlockGetValidatorsRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
blockNumber	Long	Required, the height of the block to be queried must be greater than 0

The response data is shown in the following table:

Parameter	Type	Description
validators	ValidatorInfo[]	Validators list

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_BLOCKNUMBER_ERROR	11060	BlockNumber must be greater than 0
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
BlockGetValidatorsRequest request = new BlockGetValidatorsRequest();
request.setBlockNumber(629743L);

// Call the getValidators interface
BlockGetValidatorsResponse response = sdk.getBlockService().getValidators(request);
if (response.getErrorCode() == 0) {
BlockGetValidatorsResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.6.1 ValidatorInfo

The specific information of ValidatorInfo is shown in the following table:

Member	Type	Description
address	String	Consensus node address
plegeCoinAmount	Long	Validators' deposit

2.10.7 getLatestValidators

The `getLatestValidators` interface is used to get the number of all validators in the latest block.

The method call is as follows:

```
BlockGetLatestValidatorsResponse getLatestValidators();
```

The response data is shown in the following table:

Parameter	Type	Description
validators	ValidatorInfo[]	Validators list

The error code is shown in the following table:

Exception	Error Code	Description
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Call the getLatestValidators interface
BlockGetLatestValidatorsResponse response = sdk.getBlockService().
↪getLatestValidators();
if (response.getErrorCode() == 0) {
BlockGetLatestValidatorsResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.8 getReward

The `getReward` interface is used to retrieve the block reward and valicator node rewards in the specified block. The method call is as follows:

```
BlockGetRewardResponse getReward(BlockGetRewardRequest);
```

The request parameters are shown in the following table:

Parameter	Type	Description
blockNumber	Long	Required, the height of the block to be queried must be greater than 0

The response data is shown in the following table:

Parameter	Type	Description
blockReward	Long	Block rewards
validatorsReward	ValidatorReward[]	Validators rewards

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_BLOCKNUMBER_ERROR	11060	BlockNumber must be greater than 0
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
BlockGetRewardRequest request = new BlockGetRewardRequest();
request.setBlockNumber(629743L);

// Call the getReward interface
BlockGetRewardResponse response = sdk.getBlockService().getReward(request);
if (response.getErrorCode() == 0) {
BlockGetRewardResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.8.1 ValidatorReward

The specific information of ValidatorReward is shown in the following table:

Member	Type	Description
validator	String	Validator address
reward	Long	Validator reward

2.10.9 getLatestReward

The `getLatestReward` interface gets the block rewards and validator rewards in the latest block. The method call is as follows:

```
BlockGetLatestRewardResponse getLatestReward();
```

The response data is shown in the following table:

Parameter	Type	Description
blockReward	Long	Block rewards
validatorsReward	*ValidatorReward*[]	Validator rewards

The error code is shown in the following table:

Exception	Error Code	Description
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Call the getLatestReward interface
BlockGetLatestRewardResponse response = sdk.getBlockService().getLatestReward();
if (response.getErrorCode() == 0) {
BlockGetLatestRewardResult result = response.getResult();
System.out.println(JSON.toJSONString(result, true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.10 getFees

The `getFees` interface gets the minimum asset limit and fuel price of the account in the specified block.

The method call is as follows:

```
BlockGetFeesResponse getFees(BlockGetFeesRequest);
```

The request parameter is shown in the following table:

Parameter	Type	Description
blockNumber	Long	Required, the height of the block to be queried must be greater than 0

The response data is shown in the following table:

Parameter	Type	Description
fees	Fees	Fees

The error code is shown in the following table:

Exception	Error Code	Description
INVALID_BLOCKNUMBER_ERROR	11060	BlockNumber must be greater than 0
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Initialize request parameters
BlockGetFeesRequest request = new BlockGetFeesRequest();
request.setBlockNumber(629743L);

// Call the getFees interface
BlockGetFeesResponse response = sdk.getBlockService().getFees(request);
if (response.getErrorCode() == 0) {
System.out.println(JSON.toJSONString(response.getResult(), true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.10.10.1 Fees

The specific information of Fees is shown in the following table:

Member	Type	Description
baseReserve	Long	Minimum asset limit for the account
gasPrice	Long	Transaction fuel price, unit MO, 1 BU = 10^8 MO

2.10.11 getLatestFees

The `getLatestFees` interface is used to obtain the minimum asset limit and fuel price of the account in the latest block.

The method call is as follows:

```
BlockGetLatestFeesResponse getLatestFees();
```

The response data is shown in the following table:

Parameter	Type	Description
fees	Fees	Fees

The error code is shown in the following table:

Exception	Error Code	Description
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
SYSTEM_ERROR	20000	System error

The specific example is as follows:

```
// Call the getLatestFees interface
BlockGetLatestFeesResponse response = sdk.getBlockService().getLatestFees();
if (response.getErrorCode() == 0) {
System.out.println(JSON.toJSONString(response.getResult(), true));
} else {
System.out.println("error: " + response.getErrorDesc());
}
```

2.11 Error Code

The following table describes the error messages that may appear.

Exception	Error code	Description
ACCOUNT_CREATE_ERROR	11001	Failed to create the account
INVALID_SOURCEADDRESS _ERROR	11002	Invalid sourceAddress
INVALID_DESTADDRESS_E RROR	11003	Invalid destAddress
INVALID_INITBALANCE_E RROR	11004	InitBalance must be between 1 and Long.MAX_
SOURCEADDRESS_EQUAL_D ESTADDRESS_ERROR	11005	SourceAddress cannot be equal to destAddress
INVALID_ADDRESS_ERROR	11006	Invalid address
CONNECTNETWORK_ERROR	11007	Failed to connect to the network
INVALID_ISSUE_AMOUNT _ERROR	11008	Amount of the token to be issued must be betwe
NO_ASSET_ERROR	11009	The account does not have the asset
NO_METADATA_ERROR	11010	The account does not have the metadata
INVALID_DATAKEY_ERROR	11011	The length of key must be between 1 and 1024
INVALID_DATAVALUE_ERR OR	11012	The length of value must be between 0 and 2560
INVALID_DATAVERSION_E RROR	11013	The version must be equal to or greater than 0
INVALID_MASTERWEIGHT _ERROR	11015	MasterWeight must be between 0 and (Integer.M
INVALID_SIGNER_ADDRES S_ERROR	11016	Invalid signer address
INVALID_SIGNER_WEIGHT _ERROR	11017	Signer weight must be between 0 and (Integer.M
INVALID_TX_THRESHOLD _ERROR	11018	TxThreshold must be between 0 and Long.MAX
INVALID_OPERATION_TYP E_ERROR	11019	Operation type must be between 1 and 100
INVALID_TYPE_THRESHOL D_ERROR	11020	TypeThreshold must be between 0 and Long.MA
INVALID_ASSET_CODE_ER ROR	11023	The length of key must be between 1 and 64

Table 2 – continued from previous page

Exception	Error code	Description
INVALID_ASSET_AMOUNT _ERROR	11024	AssetAmount must be between 0 and Long.MAX
INVALID_BU_AMOUNT_ERR OR	11026	BuAmount must be between 0 and Long.MAX_
INVALID_ISSUER_ADDRES S_ERROR	11027	Invalid issuer address
NO_SUCH_TOKEN_ERROR	11030	No such token
INVALID_TOKEN_NAME_ER ROR	11031	The length of token name must be between 1 and
INVALID_TOKEN_SIMBOL _ERROR	11032	The length of symbol must be between 1 and 102
INVALID_TOKEN_DECIMAL S_ERROR	11033	Decimals must be between 0 and 8
INVALID_TOKEN_TOTALSU PPLY_ERROR	11034	TotalSupply must be between 1 and Long.MAX_
INVALID_TOKENOWNER_ER RPR	11035	Invalid token owner
INVALID_CONTRACTADDRE SS_ERROR	11037	Invalid contract address
CONTRACTADDRESS_NOT_C ONTRACTACCOUNT_ERROR	11038	contractAddress is not a contract account
INVALID_TOKEN_AMOUNT _ERROR	11039	TokenAmount must be between 1 and Long.MA
SOURCEADDRESS_EQUAL_C ONTRACTADDRESS_ERROR	11040	SourceAddress cannot be equal to contractAddre
INVALID_FROMADDRESS_E RROR	11041	Invalid fromAddress
FROMADDRESS_EQUAL_DES TADDRESS_ERROR	11042	FromAddress cannot be equal to destAddress
INVALID_SPENDER_ERROR	11043	Invalid spender
PAYLOAD_EMPTY_ERROR	11044	Payload cannot be empty
INVALID_LOG_TOPIC_ERR OR	11045	The length of one log topic must be between 1 ar
INVALID_LOG_DATA_ERRO R	11046	The length of one piece of log data must be betw
INVALID_CONTRACT_TYPE _ERROR	11047	Invalid contract type
INVALID_NONCE_ERROR	11048	Nonce must be between 1 and Long.MAX_VAL
INVALID_GASPRICE_ERRO R	11049	GasPrice must be between 1000 and Long.MAX_
INVALID_FEELIMIT_ERRO R	11050	FeeLimit must be between 1 and Long.MAX_VA
OPERATIONS_EMPTY_ERRO R	11051	Operations cannot be empty
INVALID_CEILLEDGERSEQ _ERROR	11052	CeilLedgerSeq must be equal to or greater than 0
OPERATIONS_ONE_ERROR	11053	One of the operations cannot be resolved
INVALID_SIGNATURENUMB ER_ERROR	11054	SignagureNumber must be between 1 and Intege
INVALID_HASH_ERROR	11055	Invalid transaction hash
INVALID_BLOB_ERROR	11056	Invalid blob
PRIVATEKEY_NULL_ERROR	11057	PrivateKeys cannot be empty
PRIVATEKEY_ONE_ERROR	11058	One of privateKeys is invalid
PUBLICKEY_NULL_ERROR	11061	PublicKey cannot be empty
URL_EMPTY_ERROR	11062	Url cannot be empty
CONTRACTADDRESS_CODE _BOTH_NULL_ERROR	11063	ContractAddress and code cannot be empty at th
INVALID_OPTTYPE_ERROR	11064	OptType must be between 0 and 2
GET_ALLOWANCE_ERROR	11065	Failed to get allowance
GET_TOKEN_INFO_ERROR	11066	Failed to get the token info
SIGNATURE_EMPTY_ERROR	11067	The signatures cannot be empty
REQUEST_NULL_ERROR	12001	Request parameter cannot be null
CONNECTN_BLOCKCHAIN_E RROR	19999	Failed to connect to the blockchain
SYSTEM_ERROR	20000	System error

CHAPTER 3

API Guide for Exchanges

3.1 Overview

This document is for exchanges to install the BUMO node, and use the BUMO SDK.

3.2 BUMO Node Installation

Support for most operating systems such as Ubuntu, Centos, etc. It is recommended to use Ubuntu 14.04 or Centos 7. This example uses Ubuntu 14.04 as an example of a node installation.

3.2.1 Installing Dependencies

You need to install the dependencies required by the system before compiling the source code for BUMO. To install dependencies, you need to complete the following steps:

1. Input the following command to install `automake`.

```
sudo apt-get install automake
```

2. Input the following command to install `autoconf`.

```
sudo apt-get install autoconf
```

3. Input the following command to install `libtool`.

```
sudo apt-get install libtool
```

4. Input the following command to install `g++`.

```
sudo apt-get install g++
```

5. Input the following command to install `libssl-dev`.

```
sudo apt-get install libssl-dev
```

6. Input the following command to install `cmake`.

```
sudo apt-get install cmake
```

7. Input the following command to install `libbz2-dev`.

```
sudo apt-get install libbz2-dev
```

8. Input the following command to install `python`.

```
sudo apt-get install python
```

9. Input the following command to install `unzip`.

```
sudo apt-get install unzip
```

3.2.2 Compiling Source Code

The source code of BUMO can be compiled after the dependencies are successfully installed. If `git` is not installed, you can install `git` with the `sudo apt-get install git` command. To compile the source code of BUMO, you need to complete the following steps:

1. Download the BUMO source code file by inputting the following command in the root directory.

```
git clone https://github.com/bumoproject/bumo.git
```

```
root@ubuntu:/# git clone https://github.com/bumoproject/bumo.git
Cloning into 'bumo'...
remote: Counting objects: 22085, done.
remote: Compressing objects: 100% (265/265), done.
remote: Total 22085 (delta 200), reused 232 (delta 87), pack-reused 21695
Receiving objects: 100% (22085/22085), 185.05 MiB | 432.00 KiB/s, done.
Resolving deltas: 100% (10447/10447), done.
Checking connectivity... done.
Checking out files: 100% (13744/13744), done.
```

Note: The bumo/ directory will be created automatically during the BUMO source code being downloaded, and the source code files will be stored in this directory.

2. Input the following command to enter the directory where the source code files are located.

```
cd /bumo/build/
```

3. Input the following command to download the dependencies and initialize the development environment.

```
./install-build-deps-linux.sh
```

4. Input the following command to return to the bumo/ directory.

```
cd ../
```

5. Input the following command to complete the compilation of the BUMO source code. The message below shows that the compilation is successful.

```
make
```

```
make[3]: Leaving directory `/bumo/build/linux'
/usr/bin/cmake -E cmake_progress_report /bumo/build/linux/CMakeFiles  1 2 3 4 5 6 7 8 9
[100%] Built target bumo
make[2]: Leaving directory `/bumo/build/linux'
/usr/bin/cmake -E cmake_progress_start /bumo/build/linux/CMakeFiles 0
make[1]: Leaving directory `/bumo/build/linux'
```

Note: The executable files **bumo** and **bumod** generated after compilation are stored in the /bumo/bin directory.

3.2.3 Installing the BUMO Node

The BUMO node can be installed after the compilation is completed. To install the BUMO node, you need to complete the following steps:

1. Input the following command to enter the installation directory.

```
cd /bumo/
```

2. Input the following command to complete the installation. The message below shows that the installation is successful.

```
make install
```

```
sudo mkdir -p /usr/local/buchain/coredump;
make[1]: Leaving directory `/bumo/build/linux'
```

Note:

- By default the service is installed in the /usr/local/buchain/ directory.
- After the installation is complete, you can start the bumo service with the `service bumo start` command without additional configuration.
- After installing the BUMO node, the directory structure in the buchain/ directory is as follows:

Directory	Description
bin	The directory stores the executable file (compiled bumo executable)
jslib	The directory stores the third-party `js` library
config	The configuration file directory contains: bumo.json
data	The database directory stores account ledger data
scripts	The directory stores scripts to start and stop the node
log	The directory stores logs

3.2.4 Changing the Operating Environment

You need to stop the BUMO service before changing the operating environment of BUMO. You can modify it by following the steps below:

1. Input the following command to enter the configuration file directory.

```
cd /usr/local/buchain/config/
```

Note:

The following runtime environment configuration files are provided in this directory.

- Bumo-mainnet.json (This file is the configuration file of the main network environment, which is applied in the production environment)
- bumo-testnet.json (This file is the configuration file applied in the test network environment)
- bumo-single.json (This file is the configuration file applied in the single-node debugging environment)

2. Change the configuration file (bumo.json) of the current running environment to another name, for example:

```
mv bumo.json bumoprevious.json
```

3. Change the environment configuration file to be run to bumo.json, for example:

```
mv bumo-mainnet.json bumo.json
```

Note:

- In this example, the main network environment is set to the running environment.
- After changing the operating environment, you need to clear the database to restart the bumo service.

3.3 DevOps Services

How to start, stop, query the service and the system, as well as how to clear the database are described in this section.

Starting BUMO Service

Input the following command to start the bumo service.

```
service bumo start
```

Stopping BUMO Service

Input the following command stop the bumo service.

```
service bumo stop
```

Querying BUMO Service Status

Input the following command to query the bumo service.

```
service bumo status
```

Querying System Status

Input the following command to query the detailed status of the system:

```
curl 127.0.0.1:19333/getModulesStatus
```

The following response message is received:

```
{
"glue_manager":{
"cache_topic_size":0,
"ledger_upgrade":{
"current_states":null,
"local_state":null
},
"system":{
"current_time":"2017-07-20 10:32:22", //Current system time
"process_uptime":"2017-07-20 09:35:06", //When bumo was started
"uptime":"2017-05-14 23:51:04"
},
"time":"0 ms",
"transaction_size":0
},
"keyvalue_db":Object{...},
"ledger_db":Object{...},
"ledger_manager":{
"account_count":2316, //Total accounts
"hash_type":"sha256",
"ledger_sequence":12187,
"time":"0 ms",
"tx_count":1185 //Total transactions
},
"peer_manager":Object{...},
"web server":Object{...},
```

3.3.1 Clearing Database

You need to stop the BUMO service before clearing the data. To clear the database, you need to complete the following steps:

1. Input the following command to enter the bumo service directory.

```
/usr/local/buchain/bin
```

2. Input the following command to clear the database.

```
./bumo --dropdb
```

Note: After the database is successfully cleared, you can see the information shown below.

```
[2018-07-18 18:02:08.440 - INF] <7F6CC18C18C0> main.cpp(153):Initialize db succes
sful
[2018-07-18 18:02:08.440 - INF] <7F6CC18C18C0> main.cpp(156):Drop db successfully
```

3.4 JAVA SDK Usage

The use of the JAVA SDK includes *Generating Addresses for Users to Recharge*, *Checking the Legality of Account Addresses*, and *Asset Transactions*.

3.4.1 Generating Addresses for Users to Recharge

The exchange needs to generate an address for each user to recharge. The exchange can create the user's address to recharge through `Keypair.generator()` provided in Bumo-sdk-java. The specific example is as follows:

```
/**

     * Generate an account private key, public key and address
     */
    @Test
    public void createAccount() {
        Keypair keypair = Keypair.generator();
        System.out.println(JSON.toJSONString(keypair, true));
    }
```

The return value is shown below:

```
{
    "address":"buQsG8XBSNSR5xzpRzRACfQNgxL5SMr2fenF",
    "privateKey":"privbyzj3bKdsB8SkVGmfp77JM2CZ4bsHei38GkSUHx3nHhApuGxtVvo",
    "publicKey":"b0012adc1eab24afdb266eb6f0341c893cff92aaef4dbc9e21d645f6accb48e6834b6336eb56"
}
```

3.4.2 Checking the Legality of Account Addresses

Check the validity of the account address by the code shown below.

```
/**

     * Check whether the account address is valid
     */
    @Test
    public void checkAccountAddress() {
        String address = "buQemmMwmRQY1JkcU7w3nhruoX5N3j6C29uo";
        AccountCheckValidRequest accountCheckValidRequest = new␣
↪AccountCheckValidRequest();
        accountCheckValidRequest.setAddress(address);
        AccountCheckValidResponse accountCheckValidResponse = sdk.getAccountService().
↪checkValid(accountCheckValidRequest);
        if (0 == accountCheckValidResponse.getErrorCode()) {
            System.out.println(accountCheckValidResponse.getResult().isValid());
        } else {
            System.out.println(JSON.toJSONString(accountCheckValidResponse, true));
        }
    }
```

Note:

- If the return value is true, the account address is legal.

- If the return value is false, the account address is illegal.

3.4.3 Asset Transactions

In the BUMO network, a block is generated every 10 seconds, and each transaction only needs one confirmation to get its final state. In this section, we will introduce *Detecting User Recharging*, *Withdrawing or Transferring BU by Users* and *Querying Transactions*.

3.4.3.1 Detecting User Recharging

The exchange needs to monitor block generation, and then parse the transaction records in the block to confirm the user's recharge behavior. The specific steps are as follows:

1. Make sure that the node block status is normal.
2. Analyze the transactions contained in the block (for parsing methods, see parsing transactions in the block).
3. Record the results after parsing.

Viewing the Block Status

View the block status by the code shown below.

```
/**

     * Check whether the connected node is synchronous in the blockchain
     */
    @Test
    public void checkBlockStatus() {
        BlockCheckStatusResponse response = sdk.getBlockService().checkStatus();
        System.out.println(response.getResult().getSynchronous());
    }
```

Note:

- If the return value is true, the block is normal.
- If the return value is false, the block is abnormal.

Parsing Transactions in the Block

The exchange can query the transactions in the block according to the block height, and then analyze each transaction.

Example of request:

```
/**

     * Detect user recharge operations
     * <p>
     *Detect user recharge actions by parsing transactions in the block
     */
    @Test
    public void getTransactionOfBolck() {
        Long blockNumber = 617247L;// Block 617247
```

(continues on next page)

(continued from previous page)

```
        BlockGetTransactionsRequest request = new BlockGetTransactionsRequest();
        request.setBlockNumber(blockNumber);
        BlockGetTransactionsResponse response = sdk.getBlockService().
↪getTransactions(request);
        if (0 == response.getErrorCode()) {
            System.out.println(JSON.toJSONString(response, true));
        } else {
            System.out.println("Failure\n" + JSON.toJSONString(response, true));
        }
        //Detect whether an account has recharged BU
        // Analyze transactions[n].transaction.operations[n].pay_coin.dest_address

        // Note:
        // Operations are arrays, there may be multiple transfer operations
    }
```

The response message is shown below:

```
{
    "total_count": 1,
    "transactions": [{
        "close_time": 1524467568753121,
        "error_code": 0,
        "error_desc": "",
        "hash": "89402813097402d1983c178c5ec271c6890db40c3beb9f06db71c8d52dab6c86",
        "ledger_seq": 33063,
        "signatures": [{
            "public_key":
↪"b001dbf0942450f5601e39ac1f7223e332fe0324f1f91ec16c286258caba46dd29f6ef9bf93b",
            "sign_data":
↪"668984fc7ded2dd30d87a1577f78eeb34d2198de3485be14ea66d9ca18f21aa21b2e0461ad8fedefc1abcb4221d346b404
↪"
        }],
        "transaction": {
            "fee_limit": 1000000,
            "gas_price": 1000,
            "metadata": "333133323333",
            "nonce": 25,
            "operations": [{
                "pay_coin": {
                    "amount": 3000,
                    "dest_address": "buQctxUa367fjw9jegzMVvdux5eCdEhX18ME"
                },
                "type": 7
            }],
            "source_address": "buQhP7pzmjoRsNG7AkhfNxiWd7HuYsYnLa4x"
        }
    }]
}

Details on the response message:

total_count   The total number of transactions (generally 1)
transactions  Query the transaction object in the block; the array size is the total␣
↪number of transactions in the block
|__ actual_fee    Transaction fees in MO
|__close_time     Transaction time
```

(continues on next page)

(continued from previous page)

```
|__error_code     Transaction status, 0 indicates success, otherwise, failure
|__error_desc     Transaction status information
|__hash           Transaction hash
|__ledger_seq     Block height
|__signatures     Signature information
|__public_key     Public key for the signer
|__sign_data      Signature data for the signer
|__transaction    Signature object
|__fee_limit      Minimum fee, in MO
|__gas_price      Gas price in MO
|__metadata       Metadata for the transaction
|__nonce          Transactions in the original account
|__operations     Operation objects (multiple objects supported)
|__pay_coin     Operation type: built-in token
|__amount       Amount of BU transferred, in MO
|__dest_address       Recipient address
|__type        Operation type: 7 stands for built-in token transfer
|__source_address  Source account address
```

Note:

- For how to use Bumo-sdk-java, visit the following link:

  https://github.com/bumoproject/bumo-sdk-java/tree/release2.0.0

- For the example of API guide for the exchange, visit the following link:

  https://github.com/bumoproject/bumo-sdk-java/blob/release2.0.0/examples/src/main/java/io/bumo/sdk/example/ExchangeDemo.java

3.4.3.2 Withdrawing or Transferring BU by Users

For BU withdrawal operations, refer to the transfer example provided by bumo-sdk-java as follows:

```
/**
     * Send a transaction of sending bu
     *
     * @throws Exception
     */
    @Test
    public void sendBu() throws Exception {
        // Init variable
        // The account private key to send bu
        String senderPrivateKey =
↪"privbyQCRp7DLqKtRFCqKQJr81TurTqG6UKXMMtGAmPG3abcM9XHjWvq";
        // The account address to receive bu
        String destAddress = "buQswSaKDACkrFsnP1wcVsLAUzXQsemauE";
        // The amount to be sent
        Long amount = ToBaseUnit.BU2MO("0.01");
        // The fixed write 1000L, the unit is MO
        Long gasPrice = 1000L;
        // Set up the maximum cost 0.01BU
        Long feeLimit = ToBaseUnit.BU2MO("0.01");
```

(continues on next page)

(continued from previous page)

```
        // Transaction initiation account's nonce + 1
        Long nonce = 1L;

        // Record txhash for subsequent confirmation of the real result of the␣
↪transaction.
        // After recommending five blocks, call again through txhash `Get the␣
↪transaction information
        // from the transaction Hash'(see example: getTxByHash ()) to confirm the␣
↪final result of the transaction
        String txhash = sendBu(senderPrivateKey, destAddress, amount, nonce, gasPrice,
↪ feeLimit);

    }
```

Note:

- Record the hash value of the BU withdrawal operation to view the final result of the BU withdrawal operation
- The current (2018-04-23) lowest value of gasPrice is 1000MO
- It is recommended to fill in 1000000 MO for feeLimit, which equals to 0.01BU

3.4.3.3 Querying Transactions

The final result of the BU withdrawal operation can be queried by the hash value returned when the BU withdrawal operation is initiated.

The call example is as follows:

```
/**
     * Get transaction information based on the transaction Hash
     */
    @Test
    public void getTxByHash() {
        String txHash =
↪"fba9c3f73705ca3eb865c7ec2959c30bd27534509796fd5b208b0576ab155d95";
        TransactionGetInfoRequest request = new TransactionGetInfoRequest();
        request.setHash(txHash);
        TransactionGetInfoResponse response = sdk.getTransactionService().
↪getInfo(request);
        if (0 == response.getErrorCode()) {
            System.out.println(JSON.toJSONString(response, true));
        } else {
            System.out.println("Failure\n" + JSON.toJSONString(response, true));
        }
    }
```

```
public static void queryTransactionByHash(BcQueryService queryService) {
   String txHash = "";
   TransactionHistory tx = queryService.getTransactionHistoryByHash(txHash);
   System.out.println(tx);
}
```

(continues on next page)

(continued from previous page)

```
Note:
- When the number of tx.totalCount is greater than or equal to 1, the transaction␣
↪history exists
- When tx.transactions.errorCode equals 0, it indicates that the transaction is␣
↪successful, otherwise the transaction is not successful.
- For the withdrawal operation, the exchange should pay attention to the pay_coin␣
↪operation
- Example of a complete BU withdrawal response:
{
    "total_count": 1,
    "transactions": [{
        "close_time": 1524467568753121,
        "error_code": 0,
        "error_desc": "",
        "hash": "89402813097402d1983c178c5ec271c6890db40c3beb9f06db71c8d52dab6c86",
        "ledger_seq": 33063,
        "signatures": [{
            "public_key":
↪"b001dbf0942450f5601e39ac1f7223e332fe0324f1f91ec16c286258caba46dd29f6ef9bf93b",
            "sign_data":
↪"668984fc7ded2dd30d87a1577f78eeb34d2198de3485be14ea66d9ca18f21aa21b2e0461ad8fedefc1abcb4221d346b404
↪"
        }],
        "transaction": {
            "fee_limit": 1000000,
            "gas_price": 1000,
            "metadata": "333133323333",
            "nonce": 25,
            "operations": [{
                "pay_coin": {
                    "amount": 3000,
                    "dest_address": "buQctxUa367fjw9jegzMVvdux5eCdEhX18ME"
                },
                "type": 7
            }],
            "source_address": "buQhP7pzmjoRsNG7AkhfNxiWd7HuYsYnLa4x"
        }
    }]
}

total_count   The total number of transactions (generally 1)
transactions  Query the transaction object in the block, the array size is the total␣
↪number of transactions in the block
|__ actual_fee    Transaction fees in MO
|__close_time     Transaction time
|__error_code     Transaction status, 0 indicates success, otherwise, failure
|__error_desc     Transaction status information
|__hash           Transaction hash
|__ledger_seq     Block height
|__signatures     Signature information
|__public_key     Public key for the signer
|__sign_data      Signature data for the signer
|__transaction    Signature object
|__fee_limit     Minimum fee, in MO
|__gas_price     Gas price, in MO
|__metadata      Metadata for the transaction
|__nonce         Transactions in the original account
```

(continues on next page)

(continued from previous page)

```
|__operations    Operation objects (multiple objects supported)
|__pay_coin      Operation type: built-in token
|__amount        Amount of BU transferred, in MO
|__dest_address       Recipient address
|__type         Operation type: 7 stands for built-in token transfer
|__source_address  Source account address
```

3.5 BU-Explorer

BUMO provides a blockchain data browsing tool for users to query block data.

You can visit the following links to query blockchain data:

- Testnet: http://explorer.bumotest.io
- Mainnet: http://explorer.bumo.io

3.6 BUMO Wallet

BUMO provides a full-node wallet for Windows and Mac, allowing users to manage their private keys, view BU transfers, and sign transactions offline.

You can download the BUMO wallet by the following link:

https://github.com/bumoproject/bumo-wallet/releases

3.7 FAQ

Start node in BUChain command line

Q: Do I need to start the node when using the BUChain command line?

A: No.

Are the values of gas_price and fee_limit fixed

Q: Are Gas_price fixed at 1000MO and fee_limit fixed at 1000000MO?

A: They are not fixed. But at present (2018-04-23) gas_price is 1000MO, the larger the gas_price is, the higher the priority for transactions to be packaged. The fee_limit is the maximum transaction fees for the blockchain when the transaction is initiated. If the transaction is legal, the actual fees charged are less than the fee_limit filled by the caller. (gas_price can be obtained from the result.fees.gas_price field in the query result via the following link:

http://seed1.bumo.io:16002/getLedger?with_fee=true

Transfer account balance

Q: Can I transfer all the balance from my account?

A: No. In order to prevent DDOS attacks, and prevent creating a large number of spam accounts, the activated accounts of BUMO must reserve a certain amount of BU, currently at 0.1 BU (it can be obtained from the result.fees.base_reserve field in the query result via the following link:

http://seed1.bumo.io:16002/getLedger?with_fee=true

CHAPTER 4

Installation Guide for BUMO

4.1 Overview

This document will walk you through the process of installing and configuring the BUMO node in both Linux and MacOS systems.

4.2 System Requirements

Before installing a BUMO node, you must make sure that your system meets the following conditions.

Hardware Requirements

The hardware requirements must meet the following configurations:

- **Recommended**CPU 8 cores, memory 32G, bandwidth 20M, SSD disk 500G
- **Minimum**CPU 4 cores, memory 16G, bandwidth 10M, SSD disk 500G

Software Requirements

You can choose Ubuntu, Centos or MacOS systems. The following systems are supported.

- Ubuntu 14.04
- Centos 7
- Mac OS X 10.11.4

4.3 Installing the BUMO Node in Linux

The following installation example is based on Ubuntu 14.04.

Two installation methods are supported on Linux systems: *Installing by Compilation* and *Installing with a Package*.

Note:

- The root directory in the root account is used as the installation directory in this installation document. You can choose your own installation directory
- Before installing the BUMO node, you must make sure that the device's network connection is normal

4.3.1 Installing by Compilation

Installing by Compilation means that the source code of the BUMO node is first compiled into machine code that can be recognized by the computer and then installed. Installing by Compilation consists of three parts: *Installing Dependencies*, *Compiling the BUMO Source Code*, and *Installing the BUMO Node*.

4.3.1.1 Installing Dependencies

You must install the dependencies required by the system before compiling the source code of the BUMO node. You must complete the following steps to install dependencies.

1. Input the following command to install `automake`.

```
sudo apt-get install automake
```

2. Input the following command to install `autoconf`.

```
sudo apt-get install autoconf
```

3. Input the following command to install `libtool`.

```
sudo apt-get install libtool
```

4. Input the following command to install `g++`.

```
sudo apt-get install g++
```

5. Input the following command to install `libssl-dev`.

```
sudo apt-get install libssl-dev
```

6. Input the following command to install `cmake`.

```
sudo apt-get install cmake
```

7. Input the following command to install `libbz2-dev`.

```
sudo apt-get install libbz2-dev
```

8. Input the following command to install `python`.

```
sudo apt-get install python
```

9. Input the following command to install `unzip`.

```
sudo apt-get install unzip
```

4.3.1.2 Compiling the BUMO Source Code

The source code of BUMO can be compiled after the dependencies have been successfully installed. You must complete the following steps to compile the source code:

1. In the root directory, input the following command to download the source code file of BUMO. If `git` is not installed, you can install `git` with the `sudo apt-get install git` command.

```
git clone https://github.com/bumoproject/bumo.git
```

```
root@ubuntu:/# git clone https://github.com/bumoproject/bumo.git
Cloning into 'bumo'...
remote: Counting objects: 22085, done.
remote: Compressing objects: 100% (265/265), done.
remote: Total 22085 (delta 200), reused 232 (delta 87), pack-reused 21695
Receiving objects: 100% (22085/22085), 185.05 MiB | 432.00 KiB/s, done.
Resolving deltas: 100% (10447/10447), done.
Checking connectivity... done.
Checking out files: 100% (13744/13744), done.
```

Note: The bumo/ directory will be created automatically during BUMO source code being downloaded, and the source code files will be stored in this directory.

2. Input the following command to enter the file directory of the source code.

```
cd /bumo/build/
```

3. Input the following command to download the dependencies and initialize the development environment.

```
./install-build-deps-linux.sh
```

4. Input the following command to return to the bumo/ directory.

```
cd ../
```

5. Input the following command to complete the compilation of the BUMO source code. The message below shows that the compilation is successful.

```
make
```

```
make[3]: Leaving directory `/bumo/build/linux'
/usr/bin/cmake -E cmake_progress_report /bumo/build/linux/CMakeFiles  1 2 3 4 5 6 7 8 9
[100%] Built target bumo
make[2]: Leaving directory `/bumo/build/linux'
/usr/bin/cmake -E cmake_progress_start /bumo/build/linux/CMakeFiles 0
make[1]: Leaving directory `/bumo/build/linux'
```

Note: The executable files generated after compilation are **bumo** and **bumod** which are stored in the /bumo/bin directory.

4.3.1.3 Installing the BUMO Node

The BUMO node can be installed after the compilation is finished. You must complete the following steps to install a BUMO node:

1. Input the following command to enter the installation directory.

```
cd /bumo/
```

2. Input the following command to complete the installation. The message below shows that the installation is successful.

```
make install
```

```
sudo mkdir -p /usr/local/buchain/coredump;
make[1]: Leaving directory `/bumo/build/linux'
```

Note:

- By default, the service is installed in the /usr/local/buchain/ directory.
- After the installation is finished, you can start the bumo service with the `service bumo start` command without additional configuration.
- After installing the BUMO node, the directory structure in the buchain/ directory is as follows:

Directory	Description
bin	The directory stores the executable file (compiled bumo executable)
jslib	The directory stores the third-party `js` library
config	The configuration file directory contains: bumo.json
data	The database directory stores account ledger data
scripts	The directory stores scripts to start and stop the node
log	The directory stores logs. Available after bumo is started

4.3.2 Installing with a Package

Installing with a package refers to installing the BUMO node with an installation package. Installing the BUMO node with the installation package consists of five parts: *Obtaining the Installation Package and Extracting It*, *Registering the Services*, *Modifying the Service Startup Directory*, *Setting the Boot Start*, and *Selecting the Configuration File for the Running Environment*.

4.3.2.1 Obtaining the Installation Package and Extracting It

You must complete the following steps to obtain the installation package of BUMO and extract it.

1. Input the following command to download the installation package of BUMO.

```
wget https://github.com/bumoproject/bumo/releases/download/1.0.0.7/buchain-1.0.0.7-
↪linux-x64.tar.gz
```

Note:

- If you don't have `wget` installed, you can use the `apt-get install wget` command to install `wget`.
- You can find the version you need from the https://github.com/bumoproject/bumo/releases link and then right-click the version to copy the download link.

- In this example the file is downloaded to the root directory.

2. Copy the installation package to the /usr/local/ directory by inputting the following command.

```
cp buchain-1.0.0.7-linux-x64.tar.gz /usr/local/
```

Note: The above copy operation is done in the directory where the file is downloaded. You must copy the file according to the specific download directory.

3. Input the following command to go to the /usr/local/ directory.

```
cd /usr/local/
```

4. Input the following command to extract the file.

```
tar -zxvf buchain-1.0.0.7-linux-x64.tar.gz
```

Note: After extracting the file, the buchain/ directory is generated.

4.3.2.2 Registering the Services

After extracting the file, you must register the services of bumo and bumod. You must complete the following steps to register services:

1. Input the following command to register the service of bumo.

```
ln -s /usr/local/buchain/scripts/bumo /etc/init.d/bumo
```

2. Input the following command to register the service of bumod.

```
ln -s /usr/local/buchain/scripts/bumod /etc/init.d/bumod
```

4.3.2.3 Modifying the Service Startup Directory

You must complete the following steps to modify the boot directory of bumo and bumod:

1. Open the bumo file by inputting the following command in the local/ directory.

```
vim buchain/scripts/bumo
```

2. Locate `install_dir` and change the installation directory of bumo.

```
install_dir=/usr/local/buchain
```

```
#!/bin/bash
install_dir=/usr/local/buchain
script_dir=/usr/local/buchain/scripts
```

Note: By default, the directory of `install_dir` is in the /usr/local/buchain directory; you can modify it according to the specific installation directory of bumo.

3. Press `Esc` to exit editing.
4. Input `:wq` to save the file.
5. Open the bumod file by inputting the following command in the local/ directory.

```
vim /buchain/scripts/bumod
```

6. Locate `install_dir` and change the installation directory for bumod.

```
install_dir=/usr/local/buchain
```

Note: By default, the directory of `install_dir` is in the /usr/local/buchain directory; you can modify it according to the specific installation directory of bumod.

7. Press `Esc` to exit editing.
8. Input `:wq` to save the file.

4.3.2.4 Setting the Boot Start

Setting up booting includes setting the startup level, adding startup commands, and modifying file permissions. You must complete the following steps to set up the boot:

1. Input the following command to set level 1.

```
ln -s -f /etc/init.d/bumod /etc/rc1.d/S99bumod
```

2. Input the following command to set level 2.

```
ln -s -f /etc/init.d/bumod /etc/rc2.d/S99bumod
```

3. Input the following command to set level 3.

```
ln -s -f /etc/init.d/bumod /etc/rc3.d/S99bumod
```

4. Input the following command to set level 4.

```
ln -s -f /etc/init.d/bumod /etc/rc4.d/S99bumod
```

5. Input the following command to set level 5.

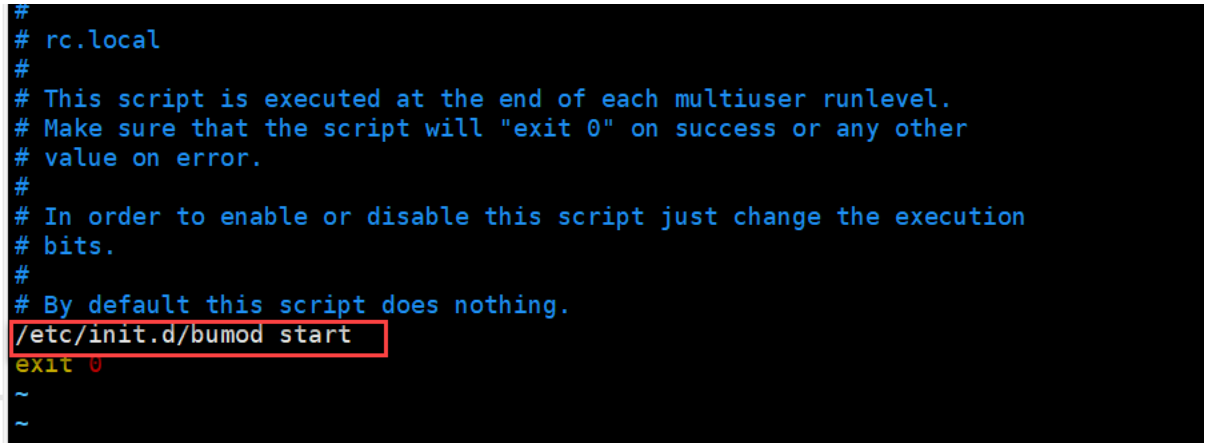
```
ln -s -f /etc/init.d/bumod /etc/rc5.d/S99bumod
```

6. Input the following command to open the rc.local file.

```
vim /etc/rc.local
```

7. Append the following command to the end of the rc.local file.

```
/etc/init.d/bumod start
```



8. Press `Esc` to exit editing.
9. Input `:wq` to save the file.
10. Execute the following command to set the permission of the rc.local file.

```
chmod +x /etc/rc.local
```

Note: Now the BUMO node is installed. Before starting the bumo service, you must select the configuration file for the running environment.

4.3.2.5 Selecting the Configuration File for the Running Environment

After installing the BUMO node, you must select the configuration file of the running environment to start the bumo service. You must complete the following steps to select the configuration file for the runtime environment:

1. Input the following command to go to the configuration file directory.

```
cd /usr/local/buchain/config/
```

Note:

The configuration files for the following runtime environments are available in this directory.

- bumo-mainnet.jsonThis file is the configuration file of the main network environment and is applied in the production environment
- bumo-testnet.jsonThis file is the configuration file of the test network environment
- bumo-single.jsonThis file is the configuration file for the single-node debugging environment

2. Input the following command to rename the configuration file for the runtime environment.

```
mv bumo-testnet.json bumo.json
```

Note:

- In this example, the test network environment is selected as the running environment. You can also select other files as your running environment according to your needs.
- After renaming the file, the bumo service can be started by the `service start bumo` command.
- After installing the BUMO node, you can view the directory structure of the installation file in the buchain/ directory.

4.4 Installing the BUMO Node in MacOS

Two installation methods are supported on MacOS systems: *Installing by Compilation in MacOS* and *Installing with a Package in MacOS*.

4.4.1 Installing by Compilation in MacOS

Installing by Compilation means that the source code of the BUMO node is first compiled into machine code that can be recognized by the computer and then installed. Installing by Compilation consists of six parts: *Installing Xcode*, *Installing Command Line Tools*, *Installing Homebrew*, *Installing Dependencies in MacOS*, *Compiling the BUMO Source Code in MacOS*, and *Installing the BUMO Node in MacOS*.

4.4.1.1 Installing Xcode

You must complete the following steps to install `Xcode`:

1. Click Software Download.
2. Input `Apple ID` and `Password`.
3. Click `Sign in` to go to the download page.
4. Click `Xcode 9.4.1` to start downloading `Xcode`.
5. Unzip the `Xcode_9.4.1.xip` file.
6. Double-click the extracted file `Xcode` to complete the installation.

Note: When choosing the version of `Xcode`, you must select one which is suitable to your MacOS system.

4.4.1.2 Installing Command Line Tools

You must complete the following steps to install `Command Line Tools`:

1. Click Software Download .
2. Input `Apple ID` and `Password`.
3. Click `Sign in` to go to the download page.
4. Click `Command Line Tools(macOS 10.14)for Xcode 10 Beta 6` to start downloading `Command Line Tools`.
5. Double-click `Command_Line_Tools_macOS_10.14_for_Xcode_10Beta_6.dmg`.
6. Click the `Command Line Tools` icon.

7. Click **Next**
8. Select a language and then click **Next**.
9. Click **Agree**.
10. Click **Install**.
11. Input password for you mac and then click **Install software**.

Note: When choosing the version of `Command Line Tools`, you must select one which is suitable to your MacOS system.

4.4.1.3 Installing Homebrew

You must complete following steps to install Homebrew:

1. Open the terminal in the MacOS system.
2. Input the following code in the terminal:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

3. Press `Enter` to install.

4.4.1.4 Installing Dependencies in MacOS

1. Input the following command to set `Homebrew` without automatic update.

```
export HOMEBREW_NO_AUTO_UPDATE=true
```

2. Input the following command to install `autoconf`.

```
brew install autoconf
```

3. Input the following command to install `automake`.

```
brew install automake
```

4. Input the following command to install `libtool`.

```
brew install libtool
```

5. Input the following command to install `cmake`.

```
brew install cmake
```

6. Input the following command to install `python`.

```
brew install python
```

7. Input the following command to install `m4`.

```
brew install m4
```

8. Input the following command to install `wget`.

```
brew install wget
```

4.4.1.5 Compiling the BUMO Source Code in MacOS

1. In the root directory, input the following command to download the source code file of BUMO. If `git` is not installed, you can install `git` with the `sudo apt-get install git` command.

```
sudo git clone https://github.com/bumoproject/bumo.git
```

Note: The bumo/ directory will be created automatically during the BUMO source code being downloaded, and the source code file will be stored in this directory.

2. Input the following command to go to the file directory of the source code.

```
cd /bumo/build/
```

3. Input the following command to download the dependencies and initialize the development environment.

```
sudo ./install-build-deps-mac.sh
```

4. Input the following command to return to the bumo/ directory.

```
cd ../
```

5. Input the following command to complete the compilation of the BUMO source code.

```
sudo make
```

Note: The executable files generated after compilation are **bumo** and **bumod** which are stored in the /bumo/bin directory.

4.4.1.6 Installing the BUMO Node in MacOS

The BUMO node can be installed after the compilation is finished. You must complete the following steps to install a BUMO node:

1. Input the following command to go to the installation directory.

```
cd /bumo/
```

2.Input the following command to complete the installation.

```
sudo make install
```

Note:

- By default, the service is installed in the /usr/local/buchain/ directory.
- After installing the BUMO node, the directory structure in the buchain/ directory is as follows:

Directory	Description
bin	The directory stores the executable file (compiled bumo executable)
jslib	The directory stores the third-party `js` library
config	The configuration file directory contains: bumo.json
data	The database directory stores account ledger data
log	The directory stores logs. Available after bumo is started

4.4.2 Installing with a Package in MacOS

Installing with a Package refers to installing the BUMO node with an installation package. Installing the BUMO node as an installation package consists of two parts: *Obtaining the Installation Package and Extracting It in MacOS*, and *Selecting the Configuration File for the Running Environment in MacOS*.

4.4.2.1 Obtaining the Installation Package and Extracting It in MacOS

1. Download the required installation package from the address below.

```
sudo wget https://github.com/bumoproject/bumo/releases/download/1.0.0.7/buchain-1.0.0.
↪7-macOS-x64.tar.gz
```

Note:

- If you don't have `wget` installed, you can use the `apt-get install wget` command to install `wget`.
- You can find the version you need from the https://github.com/bumoproject/bumo/releases link and then right-click the version to copy the download link.
- In this example the file is downloaded to the root directory.

2. Copy the installation package to the /usr/local/ directory by inputting the following command.

```
sudo cp buchain-1.0.0.7-macOS-x64.tar.gz /usr/local/
```

Note: The above copy operation is done in the directory where the file is downloaded. You must copy the file according to the specific download directory.

3. Input the following command to go to the /usr/local/ directory.

```
cd /usr/local/
```

4. Input the following command to extract the file.

```
sudo tar -zxvf buchain-1.0.0.7-macOS-x64.tar.gz
```

Note: After extracting the file, the buchain/ directory is generated.

Directory	Description
bin	The directory stores the executable file (compiled bumo executable)
jslib	The directory stores the third-party `js` library
config	The configuration file directory contains: bumo.json
data	The database directory stores account ledger data
log	The directory stores logs. Available after bumo is started

4.4.2.2 Selecting the Configuration File for the Running Environment in MacOS

After installing the BUMO node, you must select the configuration file of the running environment to start the bumo service. You must complete the following steps to select the configuration file for the runtime environment:

1. Input the following command to go to the configuration file directory.

```
cd /usr/local/buchain/config/
```

Note:

The configuration files for the following runtime environments are available in this directory.

- bumo-mainnet.jsonThis file is the configuration file of the main network environment and is applied in the production environment
- bumo-testnet.jsonThis file is the configuration file of the test network environment
- bumo-single.jsonThis file is the configuration file for the single-node debugging environment

2. Input the following command to rename the configuration file for the runtime environment.

```
mv bumo-testnet.json bumo.json
```

Note:

- In this example, the test network environment is selected as the running environment. You can also select other files as your running environment according to your needs.
- After renaming the file, the bumo service can be started by the `service start bumo` command.
- After installing the BUMO node, you can view the directory structure of the installation file in the buchain/ directory.

4.5 Configuration

The configuration is divided into *General Configuration* and *Multi-Node Configuration Example*.

4.5.1 General Configuration

General configuration includes data storage, communication between nodes, WEB API, WebSocket API, blocks, genesis, and log. The general configuration is configured in the bumo.json file in the /usr/local/buchain/config directory.

Data Storage

```
"db":{
"account_path": "data/account.db", //Store account data
"ledger_path": "data/ledger.db", //Store block data
"keyvalue_path": "data/keyvalue.db" //Store consensus data
}
```

Communication between Nodes

```
"p2p":
{
"network_id":30000,//Network ID
//Consensus network
"consensus_network":
{
"heartbeat_interval":60, //Heartbeat cycle, in second
"listen_port":36001,//Port monitored
"target_peer_connection":50, //Maximum number of active connections
"known_peers":
[
"127.0.0.1:36001"//Connect to other nodes
]
}
}
```

WEB API Configuration

```
"webserver":{
"listen_addresses":"0.0.0.0:16002"
}
```

WebSocket API Configuration

```
"wsserver":
{
"listen_address":"0.0.0.0:36003"
}
```

Block Configuration

```
"ledger":
{
"validation_address":"buQmtDED9nFcCfRkwAF4TVhg6SL1FupDNhZY",//The address of␣
↪validation node; the sync node or wallet does not need to be configured
"validation_private_key":
↪"e174929ecec818c0861aeb168ebb800f6317dae1d439ec85ac0ce4ccdb88487487c3b74a316ee777a3a7a77e5b12efd724
↪", //The private key of validation node; the sync node or wallet does not need to␣
↪be configured
```

(continues on next page)

(continued from previous page)

```
"max_trans_per_ledger":1000, //Maximum number of transactions per block
"tx_pool": //Transaction pool configuration
{
"queue_limit":10240, // Limited transactions in the transaction pool
"queue_per_account_txs_limit":64 //Maximum transaction buffer for a single account
}
}
```

Note: `Validation_address` and `validation_private_key` can be obtained through the bumo program command line tool. Please save the account information properly and you will not be able to retrieve it if it is lost.

```
[root@bumo ~]# cd /usr/local/buchain/bin
[root@bumo bin]#./bumo --create-account

{
"address" : "buQmtDED9nFcCfRkwAF4TVhg6SL1FupDNhZY", //Address
"private_key" : "privbsZozNs3q9aixZWEUzL9ft8AYph5DixN1sQccYvLs2zPsPhPK1Pt", //Private␣
↪key
"private_key_aes" :
↪"e174929ecec818c0861aeb168ebb800f6317dae1d439ec85ac0ce4ccdb88487487c3b74a316ee777a3a7a77e5b12efd724
↪", //AES encrypted private key
"public_key" :
↪"b00108d329d5ff69a70177a60bf1b68972576b35a22d99d0b9a61541ab568521db5ee817fea6", //
↪Public key
"public_key_raw" : "08d329d5ff69a70177a60bf1b68972576b35a22d99d0b9a61541ab568521db5e",
↪ //Original public key
"sign_type" : "ed25519" //ed25519 encrypted
}
```

Genesis

```
"genesis":
{
"account": "buQs9npaCq9mNFZG18qu88ZcmXYqd6bqpTU3", //Genesis address
"slogan" : "a new era of value", //Slogan stored in genesis
"fees":
{
"base_reserve": 10000000, //Base reserve for the account
"gas_price": 1000 //Byte fee
},
"validators": ["buQBwe7LZYCYHfxiEGb1RE9XC9kN2qrGXWCY"] //The block list of validation␣
↪node
}
```

Note: The `genesis` configuration on the same blockchain must be consistent. `account` can be obtained by the bumo program command line tool `./bumo --create-account`. Please save the account information properly and you will not be able to retrieve it if it is lost.

Log Configuration

```
"logger":
{
```

(continues on next page)

(continued from previous page)

```
"path":"log/buchain.log", // Log directory
"dest":"FILE|STDOUT|STDERR", //Output file classification
"level":"TRACE|INFO|WARNING|ERROR|FATAL",//Log level
"time_capacity":1, //Time span, day
"size_capacity":10, //Capacity, Megabyte
"expire_days":10 //Cycle of cleaning up the log, day
}
```

4.5.2 Multi-Node Configuration Example

In this section, two verification nodes and one synchronization node are taken as examples to describe the configuration of multiple nodes in the same blockchain. The three modules p2p, ledger and genesis need to be modified.

Configuration of p2p Module

The `known_peers` of p2p must be the IP and port of other known nodes for the interconnection between nodes.

```
verification node one:
"p2p":
{
"network_id":30000,
"consensus_network":
{
"heartbeat_interval":60,
"listen_port":36001,
"target_peer_connection":50,
"known_peers":
[
"192.168.1.102:36001", //IP and port of node two
"192.168.1.103:36001" //IP and port of node three
]
}
}

verification node two:
"p2p":
{
"network_id":30000,
"consensus_network":
{
"heartbeat_interval":60,
"listen_port":36001,
"target_peer_connection":50,
"known_peers":
[
"192.168.1.101:36001", //IP and port of node one
"192.168.1.103:36001" //IP and port of node three
]
}
}

synchronization node three:
"p2p":
{
"network_id":30000,
"consensus_network":
```

(continues on next page)

(continued from previous page)

```
{
"heartbeat_interval":60,
"listen_port":36001,
"target_peer_connection":50,
"known_peers":
[
"192.168.1.101:36001", //IP and port of node one
"192.168.1.102:36001" //IP and port of node two
]
}
}
```

Configuration of Leger Module

The `validation_address` and `validation_private_key` from the ledger of verification node must match. And you must input `validation_address` of all the verification nodes into `genesis.validators`.

```
Verification node one:
"ledger":
{
"validation_address":"buQBwe7LZYCYHfxiEGb1RE9XC9kN2qrGXWCY",//The address of␣
↪verification node one; the sync node or wallet does not need to be configured
"validation_private_key":
↪"66932f19d5be465ea9e7cfcb3ea7326d81953b9f99bc39ddb437b5367937f234b866695e1aae9be4bae27317c9987f80be
↪", //The private key of verification node two; the synchronization node or wallet␣
↪does not need to be configured
"max_trans_per_ledger":1000,
"tx_pool":
{
"queue_limit":10240,
"queue_per_account_txs_limit":64
}
}

Verification node two:
"ledger":
{
"validation_address":"buQqkp5SDcsxpwWXQ2QFQbvHKnZ199HY3dHm",//The address of␣
↪verification node two; the sync node or wallet does not need to be configured
"validation_private_key":
↪"1cb0151ec2b23cb97bf94d86ee1100582f9f5fbfdfe40a69edae2d2b8711395c40c1da859ac0bc93240a8a70c4a06779e
↪", //The private key of verification node two; the synchronization node or wallet␣
↪does not need to be configured
"max_trans_per_ledger":1000,
"tx_pool":
{
"queue_limit":10240,
"queue_per_account_txs_limit":64
}
}

Verification node three:
"ledger":
{
"max_trans_per_ledger":1000,
"tx_pool":
{
```

(continues on next page)

(continued from previous page)

```
"queue_limit":10240,
"queue_per_account_txs_limit":64
}
}
```

Configuration of Genesis Module

The genesis configuration on the same blockchain must be consistent.

```
Verification note one:
"genesis":
{
"account": "buQs9npaCq9mNFZG18qu88ZcmXYqd6bqpTU3",
"slogan" : "a new era of value",
"fees":
{
"base_reserve": 10000000,
"gas_price": 1000
},
"validators": ["buQBwe7LZYCYHfxiEGb1RE9XC9kN2qrGXWCY",
↪"buQqkp5SDcsxpwWXQ2QFQbvHKnZ199HY3dHm"] //All verification node addresses need to␣
↪be configured. If there are two verification nodes, configure two addresses.
}

Verification note two:
"genesis":
{
"account": "buQs9npaCq9mNFZG18qu88ZcmXYqd6bqpTU3",
"slogan" : "a new era of value",
"fees":
{
"base_reserve": 10000000,
"gas_price": 1000
},
"validators": ["buQBwe7LZYCYHfxiEGb1RE9XC9kN2qrGXWCY",
↪"buQqkp5SDcsxpwWXQ2QFQbvHKnZ199HY3dHm"] //All verification node addresses need to␣
↪be configured. If there are two verification nodes, configure two addresses.
}

Verification note three:
"genesis":
{
"account": "buQs9npaCq9mNFZG18qu88ZcmXYqd6bqpTU3",
"slogan" : "a new era of value",
"fees":
{
"base_reserve": 10000000,
"gas_price": 1000
},
"validators": ["buQBwe7LZYCYHfxiEGb1RE9XC9kN2qrGXWCY",
↪"buQqkp5SDcsxpwWXQ2QFQbvHKnZ199HY3dHm"] //All verification node addresses need to␣
↪be configured. If there are two verification nodes, configure two addresses.
}
```

Note:

- Before running, please make sure that the initial data of each node is consistent, otherwise you will not be able to reach consensus to generate the block.
- `account`, `validation_address` can be obtained by the bumo program command line tool `./bumo --create-account`. Please save the account information properly and you will not be able to retrieve it if it is lost.

4.6 Maintenance Service

In the maintenance service, the BUMO service operations such as startup, shutdown, status query, system details query, clear database, create a hard fork, and change the running environment, are described in detail.

Starting the BUMO Service

Input the following command to start the bumo service.

```
service bumo start
```

Note: To start the bumo service in MacOS, you must enter the /usr/local/buchain/bin directory and start the bumo service with the `./bumo` command.

Stopping the BUMO Service

Input the following command to stop the bumo service.

```
service bumo stop
```

Note: In the MacOS system, you can stop the bumo service by pressing `control+c`.

Querying the BUMO Service Status

Input the following command to query the bumo service.

```
service bumo status
```

Note: No service is available in MacOS.

Querying the Detailed System Status

Input the following command to query the detailed system status.

```
curl 127.0.0.1:19333/getModulesStatus
```

The result is shown below:

```
{
 "glue_manager":{
     "cache_topic_size":0,
     "ledger_upgrade":{
         "current_states":null,
         "local_state":null
```

(continues on next page)

(continued from previous page)

```
    },
    "system":{
        "current_time":"2017-07-20 10:32:22", //Current system time
        "process_uptime":"2017-07-20 09:35:06", //When bumo is started
        "uptime":"2017-05-14 23:51:04"
    },
    "time":"0 ms",
    "transaction_size":0
},
"keyvalue_db":Object{...},
"ledger_db":Object{...},
"ledger_manager":{
    "account_count":2316,  //Total accounts
    "hash_type":"sha256",
    "ledger_sequence":12187,
    "time":"0 ms",
    "tx_count":1185   //Total transactions
},
"peer_manager":Object{...},
"web server":Object{...},
```

Note: No service is available in MacOS.

Clearing Database

You must stop the BUMO service before clearing the database. You must complete the following steps to clear the database:

1. Input the following command to enter the bumo service directory.

```
cd /usr/local/buchain/bin
```

2. Input the following command to clear the database.

```
./bumo --dropdb
```

Note: After the database is successfully cleared, you can see the information shown below.

```
[2018-07-18 18:02:08.440 - INF] <7F6CC18C18C0> main.cpp(153):Initialize db succes
sful
[2018-07-18 18:02:08.440 - INF] <7F6CC18C18C0> main.cpp(156):Drop db successfully
```

Creating a Hard Fork

You must complete the following steps to create a hard fork.

1. Create the hard fork by inputting the following command in the /usr/local directory.

```
buchain/bin/bumo --create-hardfork
```

2. Enter `y` when prompted and then press `Enter`. The message shown below indicates the hard fork is created successfully.

```
[2018-07-19 11:45:40.464 - INF] <7F2A786AA8C0> ledger_manager.cpp(324):Are you su
re to create hardfork ledger? Press y to continue.
y
[2018-07-19 11:47:05.599 - INF] <7F2A786AA8C0> ledger_manager.cpp(341):Max closed ledger seq=290524
[2018-07-19 11:47:05.599 - INF] <7F2A786AA8C0> ledger_frm.cpp(551):total reward(800000000) = total fee(0) + block reward(800000000) in ledger(290525)
[2018-07-19 11:47:05.605 - INF] <7F2A786AA8C0> ledger_manager.cpp(438):Create hard fork ledger successful, seq(290525), consensus value hash(4b9ad78065c
65aaf1280edf6129ab2da93c99c42f2bcd380b5966750ccd5d80d)
```

Note:

- After executing the above command, the new blockchain network has only one verification node.
- After executing the hard fork command, the following hash value is displayed:

```
4b9ad78065c65aaf1280edf6129ab2da93c99c42f2bcd380b5966750ccd5d80d
```

3. Input the following command to clear the consensus status data. When clearing the consensus status data, you must ensure that the bumo service is not running, otherwise it cannot be cleared.

```
buchain/bin/bumo --clear-consensus-status
```

4. Add the hash value to the bumo.json file in the /usr/local/buchain/config directory of the node or synchronization node.

```
"ledger": {
"genesis_account": "buQs9npaCq9mNFZG18qu88ZcmXYqd6bqpTU3",
"max_trans_per_ledger": 1000,
"hardfork_points" :
[
"4b9ad78065c65aaf1280edf6129ab2da93c99c42f2bcd380b5966750ccd5d80d
"
]
},
```

5. Start the node service for the configuration to take effect.

Changing the Running Environment

Before changing the running environment, you must make sure that the BUMO service is down. If you want to change the running environment of the BUMO node, you can modify it by following the steps below.

1. Input the following command to enter the directory where the configuration file is located.

```
cd /usr/local/buchain/config/
```

Note:

The configuration files for the following runtime environments are available in this directory.

- bumo-mainnet.jsonThis file is the configuration file of the main network environment and is applied in the production environment
- bumo-testnet.jsonThis file is the configuration file of the test network environment
- bumo-single.jsonThis file is the configuration file for the single-node debugging environment

2. Change the configuration file name (bumo.json) for the current running environment, for example:

```
mv bumo.json bumoprevious.json
```

3. Change the environment configuration file to run to bumo.json, for example:

```
mv bumo-mainnet.json bumo.json
```

Note:

- In this example, the main network environment is set to the running environment.
- After changing the running environment, you must clear the database to restart the bumo service.

4.7 Uninstalling the BUMO Node

Uninstalling BUMO nodes is divided into two categories, one for uninstalling the BUMO node installed by compilation and the other is for uninstalling the BUMO node installed with a package.

4.7.1 Uninstalling the BUMO Node Installed by Compilation

If you installed the BUMO node by compilation, you can uninstall the BUMO node by the following steps:

1. Input the following command to enter the BUMO installation directory.

```
cd /bumo
```

2. Input the following command to delete the BUMO node.

```
make uninstall
```

Note: Now the BUMO node is uninstalled.

4.7.2 Uninstalling the BUMO Node Installed with a Package

If you installed the BUMO node with the installation package, you can uninstall the BUMO node by the following steps:

1. Input the following command to delete the directory of the buchain.

```
sudo rm -rf /usr/local/buchain/
```

2. Input the following command to delete the soft link of bumo.

```
sudo rm -rf /etc/init.d/bumo
```

3. Input the following command to delete the soft link of bumod.

```
sudo rm -rf /etc/init.d/bumod
```

Note: Now the BUMO node is uninstalled.

CHAPTER 5

Syntax in the Smart Contract

Bumo smart contracts are written in the 'JaveScript'. In order to facilitate developers to develop a more standardized and safer contract, JSLint is used to check the syntax in smart contracts. Please refer to JSLint GitHub. When editing a contract, you first need to ensure that the contract passes the test inJSLint before it can be detected as a legal contract by the Bumo system.

The standard syntax of JSLint is described in detail on the official website. The purpose of this document is to refine the original JSLint syntax rules as a complete document, and to supplement Bumo's modified rules. The documentation will illustrate its usage with examples. For parts not mentioned in this article, please refer to the JsLint help manual.

Or you can visit the manual at this site: 127.0.0.1:36002/jslint/help.html by node servers or wallet addresses.

5.1 Detection Tool

JSLint detection tool address: JSLint syntax dection tool.

Or you can visit 127.0.0.1:36002/jslint/index.html by node servers or wallet addresses.

For error description, details will be given when you debug contract syntax in the web tool. When you input the following code:

```
"use strict";
function init(bar)
{

}
```

Error is shown below:

```
Empty block.   2.0
{
```

Cause: Blank statement block at row 2 and column 0.

Correct code is shown below:

```
"use strict";
function init(bar)
{
    return;
}
```

If the result is correct, no warning information will prompt.

5.2 Text Compression

After the contract document is written, you can use the JSMin tool to compress it. Ensure that the original document is saved because compression is an irreversible operation.

Tool address

5.3 Demo

```
"use strict";
function init(bar)
{
    /*init whatever you want*/
    return;
}

function main(input)
{
    log(input);

    //for statement
    let i;
    for (i = 0; i < 5; i += 1)
    {
        log(i);
    }

    //while statement
    let b = 10;
    while (b !== 0)
    {
        b -= 1;
        log(b);
    }

    //if statement
    let compare = 1;
    if(compare === 1)
    {
        log("it is one");
    }
    else if(compare === 2)
    {
        log("it is two");
    }
```

(continues on next page)

(continued from previous page)

```
    else
    {
        log("it is other");
    }

    //if statement
    if(compare !== 2)
    {
        log("no, different");
    }

    //switch statement
    let sw_value = 1;
    switch(sw_value)
    {
    case 1:
        log("switch 1");
        break;
    default:
        log("default");
    }

    //Number
    let my_num = Number(111);
    log(my_num);

    //String
    let my_str = String(111);
    log(my_str);

    //Boolean
    let my_bool = Boolean(111);
    log(my_bool);

    //Array
    let str_array = ["red","black"];
    log(str_array);

    //Array
    let num_array = [1,2,3,4];
    log(num_array);

    throw "this is a exception";
}
```

5.4 Rules List

- Detect the statement strictly with all source code added the 'use strict' field at the beginning
- Use 'let' to declare variables within a statement block
- Use '===' instead of '==' to judge the comparison; use '!==' instead of '!=' to compare
- A statement must end with ';'
- A statement block must be enclosed with '{}' and empty statement blocks are prohibited

- The initial variable of the 'for' loop variable needs to be declared before the conditional statement block, and a new value is assigned to it when used
- Use '+=' and '-=' to substitute '++' and '–'
- Prohibit to use keywords like 'eval', 'void' and 'this'
- Prohibit to use 'new' to create 'Number', 'String' and 'Boolean' objects, which objects can be obtained by calling their constructors
- Prohibit to create an array with array keywords
- Prohibit to use keywords like 'try' and 'catch', but you can use 'throw' to throw exceptions

```
"Array", "ArrayBuffer", "Float32Array", "Float64Array",
"Int8Array", "Int16Array", "Int32Array", "Uint8Array",
"Uint8ClampedArray", "Uint16Array", "Uint32Array"

let color = new Array(100); //Compiling error

//You can use the alternative new Array(100) statement;
let color = ["red","black"];
let arr = [1,2,3,4];
```

- Keywords prohibited to use

```
"DataView", "decodeURI", "decodeURIComponent", "encodeURI",
"encodeURIComponent", "Generator","GeneratorFunction", "Intl",
"Promise", "Proxy", "Reflect", "System", "URIError", "WeakMap",
"WeakSet", "Math", "Date"
```