BTS's Integration Guideline for Exchanges (Single Node Version)

The purpose of this document is to assist the third party exchange listing BTS.

The framework described in this article is a single node framework. Comparing to the double node framework mentioned in the other document, single node framework enjoys the advantages of saving memory, disk and synchronization time.

1. Basic Concepts

1. 1 Consensus

BTS uses the DPOS as its consensus. BTS holders vote for the block generator. A standard block interval time is 3 seconds.

1.2 Accounts

1) Unlike Bitcoin, money is saved in an account instead of an address in BTS. The exchange should publish an account so that users can deposit.

   One can use web wallet or light wallet to register a new user.

   Attention: The exchange should use wallet mode to register new user instead of account mode. Because so advanced features required by the exchange may cause problems under the account mode.

   (In official web wallet and light wallet version 171102 and above, it is defaulted to be the account mode during the first registration. Please click "Advanced" entering wallet mode.)

   Not every account is free for registration. Usually it is free for accounts with dash or numbers (e.g. "my-exchange" or "myexchange2017").

   In the page of light wallet account, a serial of number is shown below the account name, which is the intrinsic ID of this account in BTS system and will be mentioned later.

   Attention: In official web wallet and light wallet version 171102 and above, this number is no longer shown. Please go to the blockchain explorer:

   https://cryptofresh.com/ and input the account name to acquire this ID. One can also acquire this ID by inputting command line in the wallet after the sync of self-generated node. The command line can be referred in chapter "Check of the beneficiary account of withdrawal".

   For a security purpose, exchange may use one account for deposit and the other one for withdrawal.
2) User deposit: transfer from any other accounts to the public account of the exchange

The name of the account is the beneficiary address.

Each transfer can be made with a memo, from which the user can be distinguished by the exchange.

The specific connection between memos and users should be made by the exchange. The memo is encrypted and cannot be unencrypted without the memo permission of sender or receiver.

3) User withdrawal: transfer from exchange account to user account. Beneficiary account is provided by the user. Since the user might transfer from one exchange to another, and the transfer is booked based on memo at the beneficiary exchange account, the withdrawal function is better to have memo feature.

4) The user registered via web wallet holds a basic account which can be upgraded to a lifetime membership account by paying membership fee. After upgrading, successive transaction fee is deducted by 80%.

   Membership account can create new account

   The rate of present transaction fee can be checked in the wallet. Click "Overview-Transaction Fee".

5) Each user has 3 pairs of keys, which can be checked from "Account-Permission". They includes Active Permission, Owner Permission and Memo key.

   Active Permission is used for common operations such as transfer.
   Owner Permission is used for changing password.
   Memo Key is used for encrypting and decoding the memos.

   By default, Active permission is identical to the Memo Key but can be modified. All those three keys can be modified. The Owner Permission has the highest authority which is able to modify all keys. Active Permission cannot modify the Owner Permission but can modify the other two keys.

1.3 Asset

1) There are many kinds of assets in BTS system. The core asset is BTS. Listing different assets inside the BTS is similar to listing BTS.

2) Every account may hold different assets simultaneously.

1.4 Blockchain Structure

Every block has an ID, i.e. block_id, which stores the Hash value of the block content. Every block contains the ID of the former block which is stored in "previous" field, from which the chain is built. Every block contains multiple transactions which are stored in "transaction" field sequentially.

When acquiring the block information via API, a “transaction_id” field is also returned. This is the transaction ID list which is the sequential Hash value of the transaction (signature excluded).

Each transaction may contain multiple operations which is stored in “operations” field sequentially.

Each operation has an ID too. This ID is not a Hash value but a global numbering generated along with the program running.

2. Hardware requirement

Standalone Server or VPS
8G Memory (The more the better)
50G Disk
64-bit Ubuntu 16.04 LTS (recommended)
or Ubuntu 14.04 LTS / Windows Server

3. Application preparation

Integrating BTS system requires: ordinary node witness_node, command line wallet cli_wallet

3.1 Architecture Diagram

witness_node connects to BTS network via P2P and receive the latest block from the network, then broadcast locally signed transection packages to the network.
Witness_node provides API (called node API in the following) to be called by other functions with websocket + http rpc method.

cli_wallet connects to witness_node via websocket.
Cli_wallet manages the wallet file. Wallet file contains user’s private encrypted keys. One wallet file may contain multiple keys.
Multiple cli_wallet processes can run and connect to witness_node at the same time and manage multiple wallet files simultaneously.
cli_wallet enables transaction signature which is broadcasted via witness_node after being signed.
cli_wallet provide API called by other programs(called wallet API in the following) via http rpc method.

It is recommended for the exchange to have one cli_wallet monitoring user deposit and the other cli_wllet handling the withdrawal.

3.2 Windows

.exe file is provided on Github. Download link: https://github.com/bitshares/bitshares-core/releases/latest. The file is called BitShares-Core-2.0.xxxxxx-x64-cli-tools.zip. Unzip the downloaded file gives three .exe file and two .dll file.

3.3 Linux

Linux requires compile those programs by hand. It is recommended to use Ubuntu 16.04 LTS. The compile procedure shows as follows:

sudo apt-get update
sudo apt-get install autoconf cmake git libboost-all-dev libssl-dev doxygen g++ libcurl4-openssl-dev

git clone https://github.com/bitshares/bitshares-core.git
cd bitshares-core
git checkout <LATEST_RELEASE_TAG>
git submodule update --init --recursive
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make witness_node cli_wallet

Note: In above procedure, please substitute <LATEST_RELEASE_TAG> with the latest published version serial number. It should be 2.0.171105a when this document is written.

When the compile is done, two exe files will be generated:

* build/programs/witness_node/witness_node
* build/programs/cli_wallet/cli_wallet

Those program can be copied to another directory or servers for execution. Here we assume the it stays in the present directory.

NOTE: When copying to other servers, the difference of operating system or soft/hardware environment may cause failure of the execution.

Ubuntu 14.04 LTS requires pre-installing Boost library before carrying out above procedure. Please note that currently only Boost library from 1.57.0 to 1.60.0 is supported.
The compiling and installing Boost library is as below:

sudo apt-get install cmake make libbz2-dev libdb++-dev libdb-dev libssl-dev openssl libreadline-dev autoconf libtool git autotools-dev build-essential g++ libbz2-dev libicu-dev python-dev doxygen

wget -c 'http://sourceforge.net/projects/boost/files/boost/1.57.0/boost_1_57_0.tar.bz2/download' -O boost_1_57_0.tar.bz2

```
tar xjf boost_1_57_0.tar.bz2
cd boost_1_57_0
./bootstrap.sh
sudo ./b2 install
```

Other released Linux version is also supported but it is beyond the scope of this document.

4. Environment Setting up

Ensuring stable system operation requires correct server system time. Incorrect time may lead to problems such as blockchain cannot sync or money deliver failure.

It is recommended to use NTP server for Ubuntu system. Follow the procedure below:

```
sudo timedatectl set-ntp false
sudo apt-get install ntp
```

According to the deployment it might be necessary to change the default NTP server address. Please refer to related documents for further explanation.

In Windows system, please setup the system time sync properly.

5. Data synchronization

We recommend running the program in screen or tmux on Ubuntu due to the need of running multiple programs at the same time.

Here below the description is mainly about Ubuntu, thus there is ./ in front of each command line. For Windows, after cd to the program directory in command line tool, it is no longer necessary to add ./ for execution.

5.1 witness_node

./witness_node –help can be used to check the command parameters.

5.1.1 Initial execution

```
./witness_node -d witness_node_data_dir
```

And press Ctrl+C to end it.
This operation can generate a data log file witness_node_data_dir in present directory, which contains blockchain log as data storage and config.ini configuration file.

For exchange, some modifications are recommended to be made on config.ini file.

1) Close p2p daily log to lessen the storage pressure for disk, which can be done by finding *filename=logs/p2p/p2p/log* and add "**#"** in front of the line or change *level=info* into *level=error* in [*logger.p2p*]

2) Consider saving the console log to file by following the method below:

[logger.default]
level=info
appenders=stderr

Change this into:

[log.file_appender.default]
filename=logs/default/default.log

[logger.default]
level=info
appenders=stderr,default

Afterwards, under *witness_node_data_dir/logs/default/* will have the latest 24hrs console log saved synchronously.

3) Modifying parameters below can reduce the memory required for running the system. The principle is to abandoning the historical transaction records index of BTS intrinsic transaction engine since those are usually not needed for the exchange.

history-per-size = 0

For version 2.0.171105a and above, the parameter below also need to be set:

plugins = witness account_history

Note:

- By default, there is a "#" in front of plugins in config.ini. This "#" should be deleted.
- The default configuration of plugins is "witness account_history market_history", here we actually remove the "market_history".
- If this configuration cannot be found in config.ini, for example, upgraded from old version will not upgrade the existing configuration file, then,
- One may add one line at the very beginning of config.ini (Do not add at the end of the file)
- Or find another empty directory, create a config.ini file and copy it here for modification.

4) The parameters below indicates how much historical records should be kept for search under each account. The default value is 1000. For exchange, if there is a large amount of deposit and withdrawal records, it might be set to be a larger value. For example:

max-ops-per-account = 1000

Modified to:

max-ops-per-account = 1000000

will save 100,000 records. Earlier records will be deleted from the memory so that no longer possible for quick inquiry (but still recorded on the chain).

5) Modifying parameters below will reduce the memory on use dramatically. The principle is not to save historical data index that is irrelevant to the exchange account.

track-account = "1.2.12345"
partial-operations = true

Please substitute 12345 with your account ID. The "1.2." in front of the number represents the class is account.
Note: By default, there is a “#” in front of track-account which need to be deleted.

If there are multiple accounts to be monitored, use multiple track-account configuration, for example:

track-account = "1.2.12345"
track-account = "1.2.12346"
partial-operations = true

Note:
An existing bug may lead to a failure of log modification when configure with multiple track-account. The way to overcome it is that, instead of change config.ini, adding --track-account parameters at the end of command line when booting witness_node. E.g.

./witness_node --track-account "\"1.2.12345\"" --track-account "\"1.2.12346\""

Note:

- The double quotation mark should be kept at the front and back end of the parameters so that we use \ to escape. In Linux, we can use double quotations with single quotation added at outside, such that the escape can be avoided.
- If we need add, modify, delete account tracking, then the index should be rebuilt to be effective after modification is done. Rebuilding the index can be achieved by pressing Ctrl+C to end the program, adding --replay-blockchain parameter then rebooting. E.g.

./witness_node -d witness_node_data_dir --track-account "\"1.2.12345\"" --track-account "\"1.2.12346\"" --replay-blockchain

5.1.2 Execute again

Reactivate witness_node and start sync data. The first synchronization may take a few hours or days depends on the network condition and server hardware condition.

./witness_node -d witness_node_data_dir --rpc-endpoint 127.0.0.1:8090 --track-account "\"1.2.12345\"" --track-account "\"1.2.12346\"" --partial-operations true --max-ops-per-account 1000000 --replay-blockchain

In the above command, we use –rpc-endpoint to open node API service, so that we can connect other programs with cli_wallet.

Note: Reactive witness_node in the future does not need –replay-blockchain parameter, otherwise the speed would be slow.

5.2 Execut cli_wallet for withdrawal

./cli_wallet -w wallet_for_withdrawal.json -s ws://127.0.0.1:8090 -H 127.0.0.1:8091

The above command use –w parameter indicating wallet file, -s parameter linked to witness_node, -H parameter open the wallet API service, listen port 8091.

Note:
We can use ./cli_wallet –help to check the command parameters
The data transferred between cli_wallet and witness_node does not contain encrypted data. Usually it is not necessary to encrypt or add specific protection for the node RPC port (although it is ok to add a protection layer).
However, the communication between cli_wallet and deposit and withdrawal program is plaintext and might contains password. If the deployment structure includes multiple machines, encryption should be considered. One possible method is SSH Tunnel.
Moreover, when cli_wallet is under unencrypted state, the account balance can be transferred via RPC port. Therefore we should be careful about unauthorized access. We strongly suggest that the wallet RPC should not be open to public access.
Configure certificate or password for the RPC port of cli_wallet is out of my research thus no more discussion here.

After successful execution it will show:

new >>>

First we need to set up a password for the wallet file:

new >>> set_password my_password_1234

After successful execution it will show:

locked >>>

then unlock the wallet:

locked >>> unlock my_password_1234

Successful unlock gives us:

unlocked >>>

Use info command to check the sync status:

```
unlocked >>> info
info
{
  "head_block_num": 17249870,
  "head_block_id": "0107364e2bf1c4ed1331ece4ad7824271e563fbb",
  "head_block_age": "23 seconds old",
  "next_maintenance_time": "31 minutes in the future",
  "chain_id":
"4018d7844c78f6a6c41c6a552b898022310fc5dec06da467ee7905a8dad512c8",
  "participation": "96.87500000000000000",
  ...
}
```

5.3 Execute another cli_wallet for deposit

./cli_wallet -w wallet_for_deposit.json -s ws://127.0.0.1:8090 -H 127.0.0.1:8093

This cli_wallet can open wallet API service, listen port 8093.
Please refer to the former chapter for setting up password and unlock.

6. Account setting up

Considering the security, we can use two accounts dealing for deposit and withdrawal. Here we assume deposit-account for deposit and withdrawal-account for withdrawal.

6.1 Change the memo keys for the deposit account

In any of the above cli_wallet we can get a pair of encrypted keys by running suggest_brain_key. E.g.

```
unlocked >>> suggest_brain_key
suggest_brain_key
{
  "brain_priv_key": ".....",
  "wif_priv_key": "5JxyJx2KyDmAx5kpkMthWEpqGjzpwtGtEJigSMz5XE1AtrQaZXu",
  "pub_key": "BTS69uKRvM8dAPn8En4SCi2nMTHKXt1rWrohFbwaPwv2rAbT3XFzf"
}
```

On the account permission page of light wallet, change the memo key to the pub_key as shown above.

Note:

- Remember to back up the light wallet, otherwise the wallet may not be able to decode the memo.
- If we still need to use light wallet for transferring with memos, or reading new memos of moving in/out after the modification, we need import the above mentioned wif_priv_key into the light wallet. The import procedure can be referred in the second step of the tutorial below: http://btsabc.org/article-761-1.html. A new backup can be created afterwards.
- This method can also be applied to modify the Active Permission and Owner Permission when it is needed.

6.2 Import Memo keys of deposit account into the deposit cli_wallet

Using “unlock” command to unlock the wallet if it has been locked.

A wif_priv_key from the result of above mentioned suggest_brain_key is needed here:

unlocked >>> import_key deposit-account
5JxyJx2KyDmAx5kpkMthWEpqGjzpwtGtEJigSMz5XE1AtrQaZXu

One or two auto-generated back-up files would be created while importing cli_wallet. They can be deleted.

Press Ctrl+D to exit wallet, back up the wallet file wallet_for_deposit.json and reactivate cli_wallet.

An error will be reported while exiting. Please ignore it.

If not introducing readline library while compiling, exit with Ctrl+C.

Because the Active Permission is not imported, the deposit cli_wallet can not move the money in deposit account but can only read the history records.

6.3 get withdrawal account’s active permission from light wallet.

Please refer to http://btsabc.org/article-761-1.html

6.4 Import withdrawal account’s active permission into cli-wallet that in charge of withdrawal

unlocked >>> import_key withdrawal-account
5xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Also, a backup of the wallet file can be done.

Note: please check if the withdrawal account' active permission is the same as its memo key. If not, memo key should be imported as well, otherwise, the withdrawal transaction with memo cannot be dealt with. The command of import is:

unlocked >>> import_key withdrawal-account
5xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

7. wallet command

In cli_wallet,
*use help command can get the list of the commands and parameters
*if there is doxygen while compiling,

unlocked >>> gethelp get_account

8.wallet API

When wallet open http rpc API service, you can call **all** wallet command by http rpc

For example :

curl -d '{"jsonrpc": "2.0", "method": "get_block", "params": [1], "id": 1}'
http://127.0.0.1:8093/rpc

i.e. pass the name of command to "method", pass the list of parameters to "params" array

return :

{"id":1,"result":{"previous":"0000000000000000000000000000000000000000","timestamp":"2015-10-13T14:12:24","witness":"1.6.8","transaction_merkle_root":"0000000000000000000000000000000000000000","extensions":[],"witness_signature":"1f53542bb60f1f7a653bac70d6b1613e73b9adc952031e30e591e601dd60d493ba5c9a832e155ff0c40ea1dd53512e9f93bf65a8191497ea67d701bc2502f93af7","transactions":[],"block_id":"00000001b656820f72f6b28cda811778632d4998","signing_key":"BTS6ZQEFsPmG6jWspNDdZHkehNmPpG7gkSHkphmRZQWaJ2LrcaVSi","transaction_ids":[]}}

If it executes successfully, there will be result in the return. If not, there will be error.

Note:
The URI of HTTP PRC request is /rpc.
Enter command from wallet, the return will be beautified. When using http rpc request, the return is raw data in .json. About raw data, you should note:
*the amount is in the format of {"amount":467116432,"asset_id":"1.3.0"}, where
*using get_asset command, the specific name of the asset can be searched by asset_id. BTS's asset_id is 1.3.0, while other assets have other ids.

* “amount” is the actual amount after taking out decimal point. For example, BTS has five decimal places, the actual amount in the above example is 4671.16432.
*the format of account is 1.2.xxxx, you can get the information of account by get_account
*operation type(op) is in value format. For example, 0 represents transfer operation.

9.Handling deposit

9.1 get the block number of current “irreversible block”

Different from confirmation numbers that used by Bitcoin to reduce the possibility of reversing transaction, BTS use block number of “irreversible block” to judge if the transaction can be reversed.

The transactions in “irreversible block” and the block before that, are assured to be irreversible.

```
get_dynamic_global_properties
{
  "id": "2.1.0",
  "head_block_number": 21955727,
  ...
  "last_irreversible_block_num": 21955709
}
```

where head_block_number is the lastest block number， last_irreversible_block_num is the block number of “irreversible block”.

9.2 query deposit account’s history

Use get_relative_account_history command to query deposit account’s history, to check if there is new deposit.
Such as：

unlocked >>> get_relative_account_history deposit-account 1 100 100

unlocked >>> get_relative_account_history deposit-account 101 100 200

curl -d '{"jsonrpc": "2.0", "method": "get_relative_account_history", "params": ["deposit-account",1,100,100], "id": 1}' http://127.0.0.1:8093/rpc

Four parameters are account name, minimum number, maximum return amount, maximum number, respectively

Note:

When the maximum return amount exceeds 100, A version of cli_wallet will have a bug, which will cause the inaccuracy of the result. Thus, please avoid the limit that exceeds 100.

The return is an array, ordered by time, i.e. the latest record is in the headmost.
*if there is no new deposit, the length of the array is 0.
*if there is a new record, the N item of data is result[N], and its format might be:


```
{
  "memo":"",
  "description":"Transfer 1 BTS from a to b -- Unlock wallet to see memo.   (Fee: 0.22941
BTS)",
  "op":{
    "id":"1.11.1234567",
    "op":[
      0,
      {
        "fee":{
          "amount":22941,
          "asset_id":"1.3.0"
        },
        "from":"1.2.12345",
        "to":"1.2.45678",
        "amount":{
          "amount":100000,
          "asset_id":"1.3.0"
        },
        "memo":{
          "from":"BTS7NLcZJzqq3mvKfcqoN52ainajDckyMp5SYRgzicfbHD6u587ib",
          "to":"BTS7SakKqZ8HamkTr7FdPn9qYxYmtSh2QzFNn49CiFAkdFAvQVMg6",
          "nonce":"5333758904325274680",
          "message":"0b809fa816945342234343436651 4a153981ea"
        },
        "extensions":[
        ]
      }
    ],
    "result":[
      0,
      {
      }
    ],
    "block_num":1234567,
    "trx_in_block":7,
    "op_in_trx":0,
    "virtual_op":1234
  }
}
```

It can be seen that, the result does not include the number of each record explicitly, it should be calculated and recorded by program. Generally, it is more appropriate to reverse the order of array and handle it one by one.

Firstly, you should judge if the block in which the transaction is has been irreversible. Comparing result[N]["op"]["block_num"] with last_irreversible_block_num, if it's irreversible, continue the handling. Otherwise, jump and ignore.

result[N]["op"]["op"] is in array format. Taking the first element (result[N]["op"]["op"][0]) from array, if the result is 0, it means transfer;

Then, the "to" filed can be taken from second element, i.e. result[N]["op"]["op"][1]["to"], to judge if it is identical to deposit-account ID, which can then judge if it has transferred in;

If it has transferred in, take the "asset_id" in "amount" from second filed, result[N]["op"]["op"][1]["amount"]["asset_id"], to judge if it is the correct asset type;

Then, take "amount" in "amount", i.e. result[N]["op"]["op"][1]["amount"]["amount"], and add decimal point to get the amount of deposit;

Take the outermost filled "memo", i.e. result[N]["memo"], to get user's ID in the exchange to credit to the user's account.

result[N]["op"]["id"] is the unique ID for this amount of transfer, which can be recorded and queried.

At meanwhile, it is recommended to record the data in the result: block_num, trx_in_block, op_in_trx, which means, block number, nth transaction in the block, nth operation in the transaction, respectively.

Besides, the third party may only record transaction ID(hash), or transaction signature and do not record the operation ID or block number.
Thus, in order to make checking issues convenient, we recommend to record the transaction ID and signature that correspond to operation, the method is shown below:

According to above block_num, get the content of block by calling get_block, such as:

unlocked >>> get_block 16000000

curl -d '{"jsonrpc": "2.0", "method": "get_block", "params": [160000], "id": 1}' http://127.0.0.1:8093/rpc

If "result" is the content of block, according to above trx_in_block,
Take result["transaction_ids"][trx_in_block], which is the corresponding transaction ID;
Take result["transactions"][trx_in_block]["signatures"], which is the transaction signature, an array. Because multi-sig account may include multiple signatures when transfer.

Note:

1) wallet must be unlocked before decoding memo.
2) If it is detected that deposit memo or asset type is wrong, please note, do not return it simply, since this amount of transfer may come from other exchanges. Simply returning it will make it difficult for them to handle.
3) One block may include many amounts of deposit, which leads to the block number, trx_in_block and op_in_trx taken from “result” are identical. But virtual_op will be different, you should notice this when handling.
   It can be assured that the combination of blocknum + trx_in_block + op_in_trx + vitrual_op is unique.
   You should also notice that, currently, there is a bug about virtual_op’s data: every time, if the restart parameters are not the same and are replayed, you will find virtual_op’s value is different when research history data.
4) Because of “proposal” function, which can delay the execution, using get_block and then using trx_in_block to locate will fail to get the corresponding transaction sometimes, or get the transaction that does not corresponds to deposit operation.
   There are only a few people who use “delay execution” function, but, as this function exists theoretically, you need to notice the potential wrong handling.

10. Handling withdrawal

10.1 network status check

For the sake of security, only when witness_node network is normal, withdrawal can be handled.
Using info command to check the network status in the wallet that is in charge of withdrawal.

```
unlocked >>> info
info
{
  "head_block_num": 17249870,
  "head_block_id": "0107364e2bf1c4ed1331ece4ad7824271e563fbb",
  "head_block_age": "23 seconds old",
  "next_maintenance_time": "31 minutes in the future",
  "chain_id":
"4018d7844c78f6a6c41c6a552b898022310fc5dec06da467ee7905a8dad512c8",
  "participation": "96.87500000000000000",
  ...
}
```

The filed needed to check：

* head_block_age is better to be within 1 minute
* participation is better to be above 80， which represents the network connected with witness_node has at least 80 percent of block-generating nodes that can work normally.

Besides, if the network works normally, the difference between last_irreversible_block_num and head_block_num will not be so huge (usually, within 30). This can be taken as a reference.

10.2 Check the balance in withdrawal account

Use list_account_balances command to check if the balance of withdrawal account is sufficient. (notice the asset type and take account of transaction fee)

unlocked >>> list_account_balances withdrawal-account

Note:
1) Notice the asset type
2) Taking the transaction fee into account. Because the memo is charged by length, so the transaction fee with memo is higher than the transaction fee without memo.

10.3 check the beneficiary account of withdrawal

Use get_account_id command to check if the beneficiary account entered by the client to receive the withdrawal is valid.

locked >>> get_account_id test-123
get_account_id test-123
"1.2.96698"

locked >>> get_account_id test-124
get_account_id test-124
10 assert_exception: Assert Exception
rec && rec->name == account_name_or_id:
  {}
  th_a wallet.cpp:597 get_account

10.4 send the withdrawal

Use transfer2 command to send withdrawal transaction. Such as:

unlocked >>> transfer2 withdrawal-account to-account 100 BTS "some memo"

the parameters are: name of source account, name of beneficiary account, amount, asset type, memo

This command will sign and broadcast the transaction, and then return an array, whose first element is transaction id, second element is detailed transaction content.

Note:

1) If the asset type is BTS, the amounts maximum decimal places are 5. For other assets, the decimal places can be checked by get_asset command, in “precision” filed.
2) “transfer” command can also be used, but this will not return the transaction ID directly, and other API should be called to calculate the transaction ID out. Thus, it is not recommended.
3) Memo is usually in UTF-8 code
4) It is suggested to record related data for future query, such as transaction id, detailed transaction content in JSON format, etc.

10.5 check the withdrawal result

Use get_relative_account_history command to get withdrawal-account’s withdrawal history (please refer to “handling deposit” section). If a new record is found and the block number of this transaction is earlier than last_irreversible_block_num, this means the transaction has already been written into the block and it is irreversible.

Note: before the transaction is written into the block, it is possible to appear in get_relative_account_history, and its block number will keep changing, causing the difficulty of judging the status.
Thus, please use last_irreversible_block_num to judge the status.
According to this record’s block_num field, please use get_block command to query the detailed information.

unlocked >>> get_block 12345

In the result, there should be transaction IDs in the transaction_ids field.

It is suggested to record id (1.11.x), block_num and trx_in_block from the result of get_relative_account_history mentioned above for the future query.

10.6 about re-sending the withdrawal after failure

Under some cases, the transactions are not packed into block in time after sending.
Different from Bitcoin, there is an “expiration” filed in BTS’s transaction.
When using cli_wallet to sign the broadcasting transaction, the default value of this filed is the local host’s system time plus 2 minutes. When there are too many transactions at the local host, the expiration time will be extended.

If the transaction did not been packed into block after the network time reaching this expiration time, this transaction will be abandoned by all network nodes and will not have the possibility of being packed.

Thus, if the transaction broadcast appears but does not show in the account history, please check if local host system time is lagged first.

*If the block time that corresponds to last_irreversible_block_num has exceeded the expiration time of transaction, then, it is safe to re-send the withdrawal.
*If the transaction has appeared in the history, please check if the transaction’s block number is fixed, instead of keeping updating with the latest block number.

*If the transaction's block number keeps updating, which represents the transaction has not been packed into block, please wait patiently for been packed or expired.
*If the block number is fixed, as the time goes, last_irreversible_block_num will quickly exceeds this block number when network is normal.
*If head_block_num keeps increasing, while last_irreversible_block_num dose not, it is highly possible that witness_node enters into a short fork chain, or there are some issues of network, resulting in the transaction cannot be confirmed completely.
Under this situation, please check if witness_node has a new version to update, or please get contact with development team.
*If the re-sending still cannot be packed, it is possibly because of network exception or congestion. Since this situation is rare, please get contact with development team once met.

11. Others

* cli_wallet has parameter –daemon, after the start of parameter, it will be run in the background

* if you want to close witness_node, press Ctrl C, and wait for the program exiting automatically.

* after the program exiting normally, there is no need to rebuild index when restarting. Thus, the restart will be quicker.
* after the program exiting normally, the data category witness_node_data_dircan be packed and backup. And it can be restored when needed.

* if the program exits abnormally, it may take more time to restart as index needed to be rebuilt.

* if witness_node occurs an expectation, normally, please try to restart. If restart cannot solve the issue, please try to restart with --replay-blockchain parameter, i.e. rebuilding index manually.

* if the issue still cannot be solved, please restore from backup.
* if there is no backup, please re-synchronize, which may take longer time.

* multi-signature： BTS natively support account-level multiple signature, and has the "proposal-approval" mechanism, which can initiate multiple signature request online, and then confirm the multiple signature transaction. More detail please refer to related tutorial.

*hardware wallet: there is no hardware wallet supporting BTS currently
*cold storage: it can be implemented, while the steps are kind of complex. For example:
*at the off-line machine, start witness_node and cli_wallet and enter suggest_brain_key command to generate key pairs;
*then, use light wallet to modify the account key to the key mentioned above, then account will enter into cold storage status.
*when cold storage account is needed to use,

*the method “becoming hot wallet temporarily” can be used, i.e. importing the private key into the light wallet to use it, and changing a new key after using.
* ”complete” cold mode can also be implemented, but current cli_wallet cannot support it well. If needed, please contact BTS technical team alone.

12. related information

* graphic tutorial
  http://jc.btsabc.org/

* self-built node tutorial
  http://btsabc.org/article-477-1.html

* creating account tutorial
  http://btsabc.org/article-761-1.html

* English integration guideline
  http://docs.bitshares.org/integration/exchanges/step-by-step.html

* English API documentation
  http://docs.bitshares.org/api/index.html