

# Guide

1	Usage of Achain wallets.....	2
1.1	Achain wallets under Windows .....	2
1.2	Achain wallet under Linux .....	2
1.3	Docker wallet .....	2
1.4	Wallet synchronization .....	3
1.5	Create wallet.....	4
1.6	Create accounts.....	4
2	Get transaction history .....	5
2.1	block scanning step .....	5
2.2	ori_trx_id and result_trx_id.....	6
3	ACT transaction .....	6
3.1	Query ACT transaction.....	6
3.2	Query ACT balance .....	7
3.3	Withdraw ACT .....	8
4	Contract token transaction .....	8
4.1	Query contract transaction.....	8
4.2	Query contract token balance .....	10
4.3	Withdraw of contract token .....	11

# 1 Usage of Achain wallets

## 1.1 Achain wallets under Windows

- 1、 Official download address: <https://www.achain.com/home.html>
- 2、 Start the wallet (2 kinds of startup method: QT and command line)
  - 1、 QT method, double click the icon.
  - 2、 Command line method: First go to the directory of the Achain wallet. Then start the wallet using command:

```
Achain-c.exe --rpcuser admin --rpcpassword 123456 --httpdendpoint 127.0.0.1:8299 --server --data-dir d:\config
```

## 1.2 Achain wallet under Linux

- 1、 Official download address: [https://github.com/Achain-Dev/Achain\\_linux](https://github.com/Achain-Dev/Achain_linux)
- 2、 The code need to be compiled. Compiling method refer to document:  
[https://github.com/Achain-Dev/Achain\\_linux/blob/master/linux\\_installation\\_guide](https://github.com/Achain-Dev/Achain_linux/blob/master/linux_installation_guide)
- 3、 After compiling, start the wallet using command:

```
Achain --rpcuser admin --rpcpassword 123456 --httpdendpoint 127.0.0.1:8299
```

## 1.3 Docker wallet

- 1、 Official download address: <https://github.com/Achain-Dev/Docker>
- 2、 To start the wallet please refer to document:  
<https://github.com/Achain-Dev/Docker/blob/master/README.md>

Introduction of relevant Parameters:

- 1、 rpcuser : username to start the rpc command;
- 2、 rpcpassword :password to start the rpc command;
- 3、 httpdentpoint :ip address of the http request. (Docker wallet don't need to set this parameter)
- 4、 data-dir: storage path of the sync data.

## 1.4 Wallet synchronization

After starting the wallet, the wallet automatically connects the potential node and starts to update block information.

Do not use RPC command until the wallet synchronized to the newest block.

You can use info command to check the synchronization on.

Mainly concern three parameters:

The first is `blockchain_head_block_num`, this is the current block number.

The second is `blockchain_head_block_age`, this is the age of the current block. We generate new block every 10 seconds. Therefore if this parameter is bigger than 10 seconds, the synchronization is not over yet.

The third is `network_num_connection`, this is the current connection number. If this equals 0, the synchronization won't start. This problem need to be solved in time.

Note: During synchronization, there will be delays or errors when calling RPC command.

Please call RPC command after synchronization.

For example:

```

(wallet closed) >>> info
{
  "blockchain_head_block_num": 1140371,
  "blockchain_head_block_age": "0 second old",
  "blockchain_head_block_timestamp": "2017-12-01T09:17:00",
  "blockchain_head_block_id": "f56bf37b0f76a16c46aef0ede5ad2f8d3239006c",
  "blockchain_average_delegate_participation": "100.00 %",
  "blockchain_confirmation_requirement": 1,
  "blockchain_share_supply": "1,005,701,855.00000 ACT",
  "blockchain_blocks_left_in_round": 10,
  "blockchain_next_round_time": "at least 2 minutes in the future",
  "blockchain_next_round_timestamp": "2017-12-01T09:18:40",
  "blockchain_random_seed": "75817d1800ba89e30d96f0b4f8e306a6d3a916ab",
  "client_data_dir": "d:/config",
  "client_version": "3.1.3",
  "network_num_connections": 15,
  "network_num_connections_max": 200,
  "network_chain_downloader_running": false,
  "network_chain_downloader_blocks_remaining": null,
  "ntp_time": "2017-12-01T09:17:00",
  "ntp_time_error": "-1.57976500000000010",
  "wallet_open": false,
  "wallet_unlocked": null,
  "wallet_unlocked_until": null,
  "wallet_unlocked_until_timestamp": null,
  "wallet_last_scanned_block_timestamp": null,
  "wallet_scan_progress": null,
  "wallet_block_production_enabled": null,
  "wallet_next_block_production_time": null,
  "wallet_next_block_production_timestamp": null
}

```

## 1.5 Create wallet

Use command `wallet_create` to create wallet. User accounts are saved in this wallet.

There are two parameters: first is the wallet name. second is the wallet password.

After the creation of the wallet, the wallet is unlock automatically.

For example:

```

(wallet closed) >>> wallet_create wallet wallet01
OK

```

## 1.6 Create accounts.

There two ways for current exchange to manage users:

First: Only create a main account. You can use `sub_address` to separate user account by stitching 32bit `sub_address` after the main account. (you can choose UUID by removing “-“ )

Benefit: When transferring ACT, the sub\_address will do bookkeeping work. You don't need to collect your funds.

The sub\_address method is recommended.

RPC command: wallet\_account\_create, the parameter is the account name.

This command return the address of the account. If the return value is none or error, the account name might be registered or invalid.

After doing the work we finish the preparation of the wallet.

```
wallet01 (unlocked) >>> wallet_account_create act01  
ACTHE6bM2qet3PkBCaG6kooA2xEXz4oEHQ6s
```

Second: Create accounts for every user and manage these accounts under wallet. This need to collect funds on your own.

Note: In this way you need to turn off automatic backup or the logs grow too fast.

Command: wallet\_set\_automatic\_backups false

## 2 Get transaction history

### 2.1 block scanning step

Get transaction history by block scanning,step follows:

- 1.call RPC method blockchain\_get\_block\_count to get the header block number.
- 2.call RPC method blockchain\_get\_block to query one block. The parameter is the block number.

Note: The field "user\_transaction\_ids": []. If this field is empty, there is no transaction on this block.

You can scan the next block If not empty, then do the transaction query job. user\_transaction\_ids records the trx\_id of transaction.

For example:

```

(wallet closed) >>> blockchain_get_block 1140371
{
  "previous": "a9652f371cf938671798f18f1eb7db424e3649d8",
  "block_num": 1140371,
  "timestamp": "2017-12-01T09:17:00",
  "transaction_digest": "c8cf12fe3180ed901a58a0697a522f1217de72d04529bd255627a4ad6164f0f0",
  "next_secret_hash": "132e8e2b2180c5fa36441454430406b938a9a2d6",
  "previous_secret": "5442f8c1e384968eb9cd2833b56f1590170c2acd",
  "delegate_signature": "1f2195ef7583eb55d2bd4a2372c6b3e119b0daf11e4b3c4f82a6a2de0be21c03693132953a09ada312f3dc2441f291a5522fb18ad4320b2a48dd1b256a547de5dc",
  "user_transaction_ids": [1],
  "id": "f56bf37b0f76a16c46aef0ede5ad2f8d3239006c",
  "block_size": 166,
  "latency": 0,
  "signee_shares_issued": 0,
  "signee_fees_collected": 0,
  "signee_fees_destroyed": 0,
  "random_seed": "0000000000000000000000000000000000000000000000000000000000000000",
  "processing_time": 0
}

```

Note: `trx_id` is from the block fields `user_transaction_ids`

(Transaction id on achain browser: `trx_id` for ACT transaction, `ori_trx_id` for contract calling)

## 2.2 ori\_trx\_id and result\_trx\_id

`ori_trx_id` and `result_trx_id` only occurs on contract calling. They are not occurred in ACT transaction.

`ori_trx_id` records the origin transaction, that is, the transaction relative to ACT. Therefore, when calling contract, `ori_trx_id` only records the costs of ACT, fee and call contract cost limit. As for the contract user are calling, the `ori_trx_id` only records the calling method and parameters. It won't record the result of the contract. In addition, the result of transaction called by RPC command is `ori_trx_id`.

`result_trx_id` records the complete transaction on the block. Except the ACT transaction we described above, the `result_trx_id` also records whether the calling contract is successful and the result of the calling contracts. The transaction id gotten by scanning block is `result_trx_id`.

## 3 ACT transaction

### 3.1 Query ACT transaction

Use `trx_id` in field `user_transaction_ids` as parameter.

Call RPC command: `blockchain_get_pretty_transaction` to get the from and to account of this transaction.

`from_account`: withdraw address

`to_account` : deposit address

amount: 100,000 times of the transaction amount.

For example:

```
{
  "is_virtual": false,
  "is_confirmed": true,
  "is_market": false,
  "is_market_cancel": false,
  "trx_id": "f7b58cecabc5a0a8cd8cc8244a6141f99feb29c5",
  "block_num": 60435,
  "block_position": 0,
  "trx_type": 0,
  "ledger_entries": [{
    "from_account": "ACTAGSsqKCRkyadUtqqeMZcPXot6dhTgbHGU",
    "from_account_name": "",
    "to_account": "ACTKwXoKGtYibY6fnohvsQpaCkm5Gd12wZ94",
    "to_account_name": "",
    "amount": {
      "amount": 1000000,
      "asset_id": 0
    },
    "memo": "",
    "running_balances": []
  }
],
  1,
}
```

Then call RPC command `blockchain_get_transaction` to get if there is sub-address transaction.

The field `alp_account` is the deposit address. The figure is a transaction with sub-address. You can get sub-address from field `alp_account`.

For example:

```
transaction_id_prefix: f7b58cecabc5a0a8cd8cc8244a6141f99feb29c5
[
  "f7b58cecabc5a0a8cd8cc8244a6141f99feb29c5", {
    "trx": {
      "expiration": "2017-12-04T04:17:09",
      "alp_account": "ACTKwXoKGtYibY6fnohvsQpaCkm5Gd12wZ941234567890123456789012",
      "alp_inport_asset": {
        "amount": 1000000,
        "asset_id": 0
      },
      "operations": [{
        "type": "withdraw_op_type",
        "data": {
          "balance_id": "ACTDyHHzqDiICQCFHyPUzjZu9oqXUgQrZTbe",
          "amount": 1001000,
          "claim_input_data": ""
        }
      }
    ]
  }, {
  }
]
```

### 3.2 Query ACT balance

Use RPC command: `blockchain_list_address_balances`. The parameter is ACT address.

Return: The sum of every field balance is the balance of ACT address.

### 3.3 Withdraw ACT

Call RPC method `wallet_transfer_to_address` to withdraw ACT. Before calling this method, we should open and unlock wallet. This method need 4 parameters. The first is withdraw ACT amount. This is true transaction amount. The second is const string "ACT". The third is withdraw account name. The forth is the deposit ACT address. This method will return a transaction id, which is `trx_id`.

## 4 Contract token transaction

### 4.1 Query contract transaction

1.Call RPC command: `blockchain_get_transaction` to get the from and to account of this transaction. Use `trx_id` in field `user_transaction_ids` as parameter.

2.In field operation, if the value of field type is `transaction_op_type`, this is contract transaction. Then get the value of field `result_trx_id`. Note that this `result_trx_id` is at `trx` layer. Don't get the wrong field.

3.Next, use RPC command `blockchain_get_pretty_contract_transaction`, the parameter is `result_trx_id` we get previously. We first check the field `to_account` whether it is the right token contract id. If not, just pass it. Then we check field `reserved`. First parameter is the method the contract called. If the value is `transfer_to`, this is a contract transaction. If not, skip it. The second parameter is the parameter of method `transfer_to`, using to separate different parameters. The first is deposit address, the second is transaction amount. To monitor whether your address has received the contract token, you can check if this address is your own address.



```
wallet <locked> >>> blockchain_get_transaction fd88fb7d1b3aac1fe617f8332fdb62db047117e
[
  "fd88fb7d1b3aac1fe617f8332fdb62db047117e", {
    "trx": {
      "expiration": "2017-12-05T10:01:49",
      "alp_account": "",
      "alp_inport_asset": {
        "amount": 0,
        "asset_id": 0
      },
    },
    "operations": [ {
      "type": "transaction_op_type",
      "data": {
        "trx": {
          "expiration": "2017-12-05T10:01:49",
          "alp_account": "",
          "alp_inport_asset": {
            "amount": 0,
            "asset_id": 0
          },
        },
        "operations": [ {
          "type": "call_contract_op_type",
          "data": {
            "caller": "ACT7UZfdCJoNGzeAiEuK7dxzdtQzzhXWu5F2UmZ9fcNc2JKQLRzWk",
            "balances": [ [
              "ACT9Ek1tYy4KUNMKx49Wocm5uUJjYnSfJnY",
              2000
            ]
            ],
            "contract": "ACT92cJUUM6q$9qp1ihnJB5DJrf1pP9F2f$B",
            "costlimit": {
              "amount": 1000,
              "asset_id": 0
            },
            "transaction_fee": {
              "amount": 1000,
              "asset_id": 0
            },
          },
        },
      ],
    },
  ],
}
```

```

}
],
"result_trx_type": "complete_result_transaction",
"result_trx_id": "fd88fb7d1b3aac1fe617f8332fdb62db047117e",
"signatures": [ ]
},

```

```

wallet <locked> >>> blockchain_get_pretty_contract_transaction
transaction_id_prefix: fd88fb7d1b3aac1fe617f8332fdb62db047117e
{
  "result_trx_id": "fd88fb7d1b3aac1fe617f8332fdb62db047117e",
  "orig_trx_id": "31c7e13f79ba1b9952aae7b104020ddc89603a81",
  "block_num": 1174841,
  "block_position": 0,
  "trx_type": 14,
  "is_completed": false,
  "to_contract_ledger_entry": {
    "from_account": "ACT7bWPwBBHUFoCkmHitJ318FZ6Pn1E9gFPx",
    "from_account_name": "coinfix",
    "to_account": "CON92cJUUM6qS9qp1ihnJB5DJrf1pP9F2fSB",
    "to_account_name": "USD_COIN",
    "amount": {
      "amount": 0,
      "asset_id": 0
    },
    "fee": {
      "amount": 1660,
      "asset_id": 0
    },
    "memo": ""
  },
  "from_contract_ledger_entries": [],
  "timestamp": "2017-12-05T09:02:00",
  "expiration_timestamp": "2017-12-05T10:01:49",
  "reserved": [
    "transfer_to",
    "ACTLM5zptEYL6kqDfJG5AJUuucyu2DD75EW93df22f5790062ed839ed4f0bb395f00d1999.9900000"
  ]
}

```

## 4.2 Query contract token balance

Use RPC command `blockchain_get_events`, the parameter is the block number and `result_trx_id`.

To get `event_type` and `event_param`.

`event_type` indicate whether the transaction succeed. If the result is `transfer_to_success`, the transaction succeed. If not, the transaction failed.

`event_param` have 4 parameters, separate by “,”. First is withdraw address and balance, separate by ”:”. Second is deposit address and balance. Third is the version (increment). Forth is timestamp.

Note: balance is the balance after this transaction.

```

wallet <locked> >>> blockchain_get_events 1174841 fd88fb7d1b3aac1fe617f8332fdb62db047117e
{
  "id": "ACT92cJUUM6qS9qp1ihnJB5DJrf1pP9F2fSB",
  "event_type": "transfer_to_success",
  "event_param": "ACT7bWPwBBHUFoCkmHitJ318FZ6Pn1E9gFPx:3672000000,ACTLM5zptEYL6kqDfJG5AJUuucyu2DD75EW9:13552185687,263,1512464510",
  "is_truncated": false
}

```

### 4.3 Withdraw of contract token

The transfer function of contract token is realized by contract calling.

Therefore we should use RPC command `call_contract` to transfer contract token.

`call_contract` method needs 6 parameters:

First is `contract_id`, start with "CON".

Second is user account who calls contract, that is, withdraw account.

Third is calling method. Here we use method `transfer_to`.

Forth is parameter that calling method used. The format is `to_address|amount`. The parameters are separate by "|".

Fifth parameter is const string "ACT".

Sixth parameter is the cost limit of contract call. This can be set to any number bigger than 0.01.

Note: Transfer amount up to 5 decimal places. If decimal more than 5, the last will be ignored.

```
wallet01 <unlocked> >>> call_contract CONqfnUwosAcc3YN5D1j3PCh7G4siXPScWK act0 t
transfer_to ACTKw%oKGtYibY6fnohvsQpaCkm5Gd12wZ94!10 ACT 1
{
  "index": 0,
  "entry_id": "9fa3db6635c4feb71bd09d6ff8b267dc8b4f36e1",
  "block_num": 0,
  "is_virtual": false,
  "is_confirmed": false,
  "is_market": false,
  "trx": {
    "expiration": "2017-12-04T07:27:24",
    "alp_account": "",
    "alp_inport_asset": {
      "amount": 0,
      "asset_id": 0
    },
  },
  "operations": [{
    "type": "call_contract_op_type",
    "data": {
      "caller": "ACT74xPhBdsW5CMrcnH6rSDBdbUqN1yEd4C5MTsAYJpQhoNxU2ER",
      "balances": []
    }
  ]
}
```

After `call_contract`, the block will return the specific information of the transaction. Here we get field `entry_id`, that is `ori_trx_id` we described before.

Call RPC method `blockchain_get_contract_result`, use `entry_id` as parameter. There are two parameters in the result. The first is the block number that the transaction occurred. The second is `result_trx_id`. Then use these two parameter to call RPC method `blockchain_get_events`.

Then we will get `event_type` and `event_param`.

Among these parameters, event\_type indicate whether the transaction succeed.If the result is transfer\_to\_success, the transaction succeed. If not, the transaction failed.

There are 4 parameters in field event\_param, separate by “,”. First is withdraw address and balance, separate by “:”. Second is deposit address and balance. Third is the version (increment). Forth is timestamp.

Note: after call\_contract, the transaction is not record in the block immediately. Therefore when querying transaction information, we should wait until the new block generates or the query result may be empty. We generate new block every 10 seconds

```
lwallet01 (unlocked) >>> blockchain_get_contract_result 743585dae0a3bdd93c7b1ab0
4a6b55f10ef2f18f
{
  "block_num": 79078,
  "trx_id": "d2ad6d51dfe1d503c602512f662b31b1b1f7c6dd"
}
```

```
wallet01 (unlocked) >>> blockchain_get_events
block_number: 79078
trx_id: d2ad6d51dfe1d503c602512f662b31b1b1f7c6dd
[
  {
    "id": "ACTqfnUwosAcc3YN5D1j3PCh7G4siXPScWK",
    "event_type": "transfer_to_success",
    "event_param": "ACTAGSsqKCRkyadUtqqeMZcPKot6dhTGbHGU:99999848000000,ACTKwXoK
GtYiby6fnohvsQpaCkm5Gd12wZ94:151000000,6,1512543860",
    "is_truncated": false
  }
]
```