Protocon Network Document

wyuinche

Sep 21, 2022

PROTOCON BLOCKCHAIN

Hello! This is the document for Protocon Network.

CHAPTER

ONE

PROTOCON NETWORK

Protocon is a blockchain project that aims to build a self-operating digital economy based on protocols.

ISAAC+, Protocon's core consensus protocol, is designed for large-scale data processing for use in the industry, and has secured versatility so that it can be used in any area that requires blockchain technology. Based on this, we will promote a protocol-based pure digital economy.

For more information about Protocon, visit Protocon.

CHAPTER

TWO

CONTENTS

2.1 Mitum Protocol

2.1.1 What is MITUM?

Mitum is a general privacy blockchain that is flexible and resilient.

Mitum can be used for various kind of purposes.

- public and private blockchain like cryptocurrency network
- data-centric blockchain for arbitrary data
- secure anonymity voting system

If you want to know more about **MITUM**, visit Mitum Document.

2.1.2 Mitum Technical SPEC

- Mitum (blockchain core framework) uses *ISAAC+ consensus protocol* based on *PBFT*.
- The network transport protocol is quic (based on udp).
- *Gossip-Based* Node Discovery Protocol.
- The main storage engine of the blockchain uses *MongoDB* and the local file system is used for block storage.
- Parallel operation processing
- Main hash algorithm: Keccak 256, SHA-3
- Supports multiple hash algorithm: `Keccak 256`, `Keccak 512`, `Raw bytes.`
- Supports multiple message serialization format: *JSON*, *BSON*
- Small amount of code.
- *JSON logging*

2.2 Blockchain Application Model

Mitum is designed to be used as a general purpose blockchain. To meet this requirement, the **policy** and **data** of Mitum can be configured and managed in a practical way.

In simpler terms, network designers will design their network in two parts:

- Data
- Policy

By configuring the *data* and *policy*, designers can build and launch their own model of network.

For example, suppose that a designer wants to build a currency model in Mitum. The designer can define several currencies and relative data and add additional policy.

Data types:

- account
- balance

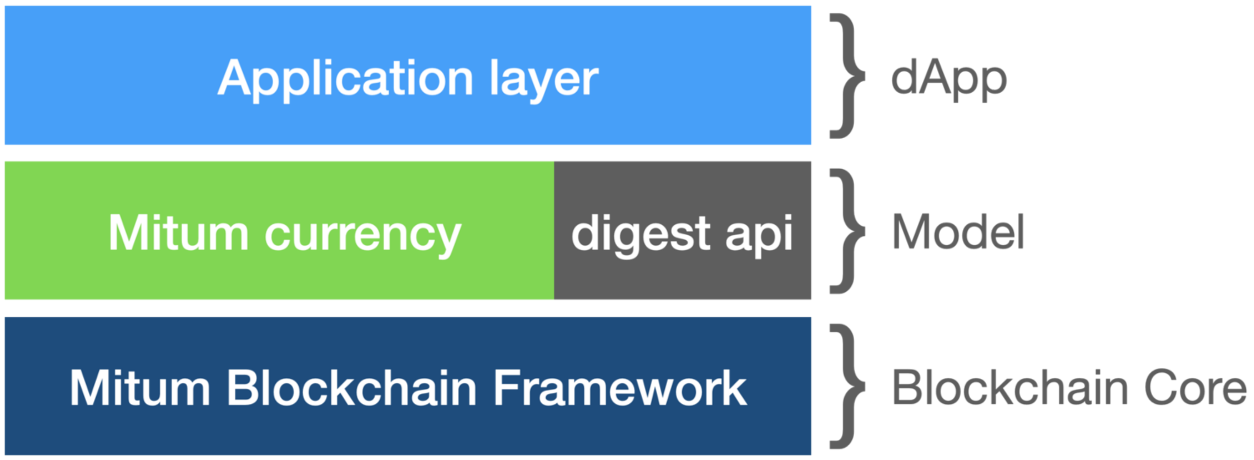
Policy:

- total amount
- minimum amount of new balance
- multisig
- inflation
- etc…

2.3 Mitum Currency

2.3.1 What is Mitum Currency

- **Mitum Currency** is a currency model that operates on the **Mitum** blockchain networks.
- The Mitum model is a solution that can provide various services as an extension layer that extends the Mitum main chain.
- In Mitum Currency, flexible policy settings related to currency issuance and operation are possible.
- Mitum Currency is implemented based on Mitum (blockchain core framework).



2.3.2 Features of Mitum Currency

- Mitum Currency provides core features to meet the business needs of various token-related fields.
- *Multiple keys* can be registered when creating an account, and related keys can be replaced through the key update operation.
- Mitum Currency can issue new currencies and related policies can be customized.
- Currency-related policy can be updated at any time as needed.
- Mitum Currency has no compensation for block generation and there is also no inflation.
- The node configuration for the Mitum Currency network follows the node operation policy of the Mitum blockchain, and details can be found at *Build Multi Nodes Network*.

2.3.3 Digest Service

- **Digest Service** is an internal service that stores block data stored by Mitum separately to serve as *HTTP-based API*.
- For more information on Digest Service, please refer to rest api.

2.3.4 Seal and Operation

Operation

In the Mitum blockchain network, an operation is **a unit of command that changes data**.

Mitum Currency has operations of,

- `create-account`
- `transfer`
- `key-updater`
- `currency-register`

- `currency-policy-updater`
- `suffrage-infration`

Each operation requires a signature made with a private key according to its contents.

The fact in the operation contains the contents to be executed and the hash value summarizing the body of the fact.

Fact and token

Every operation contains a *fact*. In other words, **the content of the operation is actually contained in the fact**.

Facts play an important role in Mitum Currency.

- The fact hash is a value representing the processed operation.
- The fact hash must have a unique value in the blockchain.
- So to check whether the operation is stored in the block, it can be retrieved using the fact hash.

In fact, the contents of the facts can be duplicated.

For example,

```
- The contents that `sender A sends 100 to receiver B` must always have the␣
↪same fact.
- Fact hashes created using the same fact content can result in duplicate␣
↪values.
- If there are two or more operations that result in duplicate values of the␣
↪fact hash, only the first operation is processed and the remaining␣
↪operations are ignored.
```

If so, does that mean that operations with the same fact content cannot be duplicated?

Don't worry, in each fact, we use a value called *token* to make it unique.
The token is a value added to the essential contents of the operation.

```
{
    "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
    "hash": "3Zdg5ZVdNFRbwX5WU7Nada3Wnx5VEgkHrDLVLkE8FMs1",
    "token": "cmFpc2VkIGJ5",
    "sender": "8PdeEpvqfyL3uZFHRZG5PS3JngYUzFFUGPvCg29C2dBnmca",
    "items": [
```

(continues on next page)

(continued from previous page)

```
        {
            "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
            "keys": {
                "_hint": "mitum-currency-keys-v0.0.1",
                "keys": [
                    {
                        "_hint": "mitum-currency-key-v0.0.1",
                        "weight": 100,
                        "key": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                    }
                ],
                "threshold": 100
            },
            "amounts": [
                {
                    "_hint": "mitum-currency-amount-v0.0.1",
                    "amount": "333",
                    "currency": "MCC"
                }
            ]
        }
    ]
}
```

The role of the token resembles that of a memo, but is capable of making a fact unique by **giving different token values** for the same fact content.

Making the fact that is essential for every operation unique expands usability in several ways.

- The biggest advantage is that you can simply check whether the operation is processed or not as you exactly know the contents of the fact along with the token.
- Anyone can calculate the fact hash if they know: the sender, receiver, currencyID, amount of currency, and a specific token value used.
- Therefore, anyone can inquire whether the corresponding operation has been processed with the fact hash.

A *fact hash* is like a **public proof** recorded in a blockchain. There can be various applications depending on how a user uses the evidence disclosed in the blockchain.
For example, even an outsider who does not have a direct account in the blockchain can check the fact hash, the only value indicating whether the operation is processed or not, and make the implementation based on that.

In addition, facts and tokens can practically be used in models that deal with various data including remittance.

Seal

Seal is **a collection of operations** transmitted to the network. In other words, the operation is contained in the seal and transmitted.

- To transmit the seal, a signature made with a private key is required.
- To create signature, you must use the private key created in Mitum's keypair package.
- Seal can contain up to 100 operations.

The private key used for the signature has nothing to do with the blockchain account. In other words, it doesn't have to be the private key used by the account.

Send

After creating an operation, the client creates and attaches a signature.

- Create as many operations as necessary within the maximum number able to be included in the seal, and put them in the seal.
- Create and put a signature on the seal.
- Send seal to Mitum node.

Stored in Block

The operation transmitted to the Blockchain network changes the state of the account if it is normal and is finally saved in the block.
Whether the operation is confirmed and saved in the block can be checked through rest api.

2.3.5 Block Data

Block data in Mitum Currency Node

In the **Mitum Currency Node**, block data is stored in two spaces: **Database** and **File System**.

- The **database** stores the information used for consensus, such as,

```
blockdata_map
info
manifest: block header
operation: operation fact
operation
proposal
seal
state: state data by each block
voteproof
```

- The **file system** stores all block data, such as,

```
manifest
operations of block
states of block
proposal
suffrage information
voteproofs(and init and accept ballots)
```

- Block data stored in the **database** is required to run the mitum currency node and participate in the network normally.
- Block data in the **file system** is not used at runtime, but is used to provide block data to syncing nodes.

An intact node must support block data for other nodes which want to synchronize block data.

BlockDataMap

By default, block data is stored in the local file system.

blockdatamap contains the information about where the actual block data is located.

```
{
    "_hint": "base-blockdatamap-v0.0.1",
    "hash": "2ojLCZwG5J7xmfoxiBbhvJsc6dDTxDFDsw1nfPneT2xr",
    "height": 2,
    "block": "BcXqCKG5MbQcfuFpPtjvHcNBGeK6Pz3aG2cMcp4MUy9C",
    "created_at": "2021-06-14T03:20:24.887Z",
    "items": {
        "operations_tree": {
            "type": "operations_tree",
            "checksum": "1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26
↪",
            "url": "file:///000/000/000/000/000/000/002/2-operations_tree-
↪1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz"
        },
    },
    "writer": "blockdata-writer-v0.0.1"
}
```

In this BlockDataMap example, the data of `operation_tree` is located at `file:///000/000/000/000/000/000/002/2-operations_tree-1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz`

BlockDataMap for block data stored in external storage

Mitum Currency supports storing block data in external storage rather than the node's local file system.

After going through a certain process to store block data externally, *blockdatamap* is modified as follows.

```
{
    "_hint": "base-blockdatamap-v0.0.1",
    "hash": "2ojLCZwG5J7xmfoxiBbhvJsc6dDTxDFDsw1nfPneT2xr",
    "height": 2,
    "block": "BcXqCKG5MbQcfuFpPtjvHcNBGeK6Pz3aG2cMcp4MUy9C",
    "created_at": "2021-06-14T03:20:24.887Z",
    "items": {
        "operations_tree": {
            "type": "operations_tree",
            "checksum": "1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26
↪",
            "url": "fhttps://aws/2-operations_tree-
↪1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz"
        },
    },
    "writer": "blockdata-writer-v0.0.1"
}
```

As you can see, the `url` is replaced with the external storage server.

How to update BlockDataMap for external Storage

For example, suppose that block data with a block height of 10 is moved to an external storage.

Here, we will do this using the node's *deploy key*.
This *deploy key* of the node is a key that can be used instead of the private key.

See `deploy key` command in *Deploy Command* for how to create a deploy key.

The process of **moving block data** and **updating blockdatamap** is as follows.

- Get the new *deploy key* of mitum currency node.
- Download the current *blockdatamap* by using the `storage download map` command.
- Upload all the block data files of height 10 to external storage(example : AWS S3)
- Update the `url` field value of the downloaded BlockDataMap with the new url of external storage.
- Update the node's *blockdatamap* by running the `storage set-blockdatamaps` command.

- Check the newly updated *blockdatamap* with `storage download map` command

After updating blockdatamap successfully, the mitum currency node will remove all the files with the height of 10 automatically after 30 minutes.

```
$ DEPLOY_KEY=d-974702df-89a7-4fd1-a742-2d66c1ead6cd

$ NODE=https://127.0.0.1:54321

$ ./mc storage download map 10 --tls-insecure --node=$NODE > mapData

$ cat mapData | jq
{
    "_hint": "base-blockdatamap-v0.0.1",
    "hash": "2ojLCZwG5J7xmfoxiBbhvJsc6dDTxDFDsw1nfPneT2xr",
    "height": 2,
    "block": "BcXqCKG5MbQcfuFpPtjvHcNBGeK6Pz3aG2cMcp4MUy9C",
    "created_at": "2021-06-14T03:20:24.887Z",
    "items": {
        "operations_tree": {
            "type": "operations_tree",
            "checksum": "1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26
↪",
            "url": "file:///000/000/000/000/000/000/002/2-operations_tree-
↪1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz"
        },
        "manifest": {
            "type": "manifest",
            "checksum": "6e53950e3ab87008b2bcb9841461588456c3e1069458eb8b150f1bfb97d22d42
↪",
            "url": "file:///000/000/000/000/000/000/002/2-manifest-
↪6e53950e3ab87008b2bcb9841461588456c3e1069458eb8b150f1bfb97d22d42.jsonld.gz"
        },
        "suffrage_info": {
            "type": "suffrage_info",
            "checksum": "e7584f9b5324566d4c5319db33ece980000f9c29eaf4d17befcc239743788f02
↪",
            "url": "file:///000/000/000/000/000/000/002/2-suffrage_info-
↪e7584f9b5324566d4c5319db33ece980000f9c29eaf4d17befcc239743788f02.jsonld.gz"
        },
        "states": {
            "type": "states",
            "checksum": "d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59
↪",
            "url": "file:///000/000/000/000/000/000/002/2-states-
↪d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59.jsonld.gz"
        },
        "operations": {
            "type": "operations",
            "checksum": "d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59
↪",
```

(continues on next page)

(continued from previous page)

```
            "url": "file:///000/000/000/000/000/000/002/2-operations-
↪d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59.jsonld.gz"
        },
        "proposal": {
            "type": "proposal",
            "checksum": "dbbce4aaa6aece06596ecd45068008d35a41f592339d8898501b55f5843dbefe
↪",
            "url": "file:///000/000/000/000/000/000/002/2-proposal-
↪dbbce4aaa6aece06596ecd45068008d35a41f592339d8898501b55f5843dbefe.jsonld.gz"
        },
        "init_voteproof": {
            "type": "init_voteproof",
            "checksum": "705af3bd660070813354b572288204d787a949fc5411f3e2bc28e86f07bc1e64
↪",
            "url": "file:///000/000/000/000/000/000/002/2-init_voteproof-
↪705af3bd660070813354b572288204d787a949fc5411f3e2bc28e86f07bc1e64.jsonld.gz"
        },
        "accept_voteproof": {
            "type": "accept_voteproof",
            "checksum": "0d4296d44f96a3de216a90f99d77bf77a00ecd5102d7bbba612b13a57bdf2f34
↪",
            "url": "file:///000/000/000/000/000/000/002/2-accept_voteproof-
↪0d4296d44f96a3de216a90f99d77bf77a00ecd5102d7bbba612b13a57bdf2f34.jsonld.gz"
        },
        "states_tree": {
            "type": "states_tree",
            "checksum": "1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26
↪",
            "url": "file:///000/000/000/000/000/000/002/2-states_tree-
↪1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz"
        }
    },
    "writer": "blockdata-writer-v0.0.1"
}

$ aws s3 cp ./blockdata/000/000/000/000/000/000/002 s3://destbucket/blockdata/000/000/
↪000/000/000/000/002 --recursive
# update mapData blockdata url from "file:///000/000/000/000/000/000/002/" to https://
↪aws/"

$ ./mc storage set-blockdatamaps $DEPLOY_KEY mapData $NODE --tls-insecure

$ ./mc storage download map 2 --tls-insecure --node=$NODE
{
    "_hint": "base-blockdatamap-v0.0.1",
    "hash": "2ojLCZwG5J7xmfoxiBbhvJsc6dDTxDFDsw1nfPneT2xr",
    "height": 2,
    "block": "BcXqCKG5MbQcfuFpPtjvHcNBGeK6Pz3aG2cMcp4MUy9C",
    "created_at": "2021-06-14T03:20:24.887Z",
    "items": {
        "operations_tree": {
            "type": "operations_tree",
```

(continues on next page)

(continued from previous page)

```
            "checksum": "1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26
↪",
            "url": "fhttps://aws/2-operations_tree-
↪1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz"
        },
        "manifest": {
            "type": "manifest",
            "checksum": "6e53950e3ab87008b2bcb9841461588456c3e1069458eb8b150f1bfb97d22d42
↪",
            "url": "fhttps://aws/2-manifest-
↪6e53950e3ab87008b2bcb9841461588456c3e1069458eb8b150f1bfb97d22d42.jsonld.gz"
        },
        "suffrage_info": {
            "type": "suffrage_info",
            "checksum": "e7584f9b5324566d4c5319db33ece980000f9c29eaf4d17befcc239743788f02
↪",
            "url": "fhttps://aws/2-suffrage_info-
↪e7584f9b5324566d4c5319db33ece980000f9c29eaf4d17befcc239743788f02.jsonld.gz"
        },
        "states": {
            "type": "states",
            "checksum": "d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59
↪",
            "url": "fhttps://aws/2-states-
↪d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59.jsonld.gz"
        },
        "operations": {
            "type": "operations",
            "checksum": "d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59
↪",
            "url": "fhttps://aws/2-operations-
↪d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59.jsonld.gz"
        },
        "proposal": {
            "type": "proposal",
            "checksum": "dbbce4aaa6aece06596ecd45068008d35a41f592339d8898501b55f5843dbefe
↪",
            "url": "fhttps://aws/2-proposal-
↪dbbce4aaa6aece06596ecd45068008d35a41f592339d8898501b55f5843dbefe.jsonld.gz"
        },
        "init_voteproof": {
            "type": "init_voteproof",
            "checksum": "705af3bd660070813354b572288204d787a949fc5411f3e2bc28e86f07bc1e64
↪",
            "url": "fhttps://aws/2-init_voteproof-
↪705af3bd660070813354b572288204d787a949fc5411f3e2bc28e86f07bc1e64.jsonld.gz"
        },
        "accept_voteproof": {
            "type": "accept_voteproof",
            "checksum": "0d4296d44f96a3de216a90f99d77bf77a00ecd5102d7bbba612b13a57bdf2f34
↪",
            "url": "fhttps://aws/2-accept_voteproof-
```

(continues on next page)

(continued from previous page)

```
↪0d4296d44f96a3de216a90f99d77bf77a00ecd5102d7bbba612b13a57bdf2f34.jsonld.gz"
        },
        "states_tree": {
            "type": "states_tree",
            "checksum": "1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26
↪",
            "url": "fhttps://aws/2-states_tree-
↪1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz"
        }
    },
    "writer": "blockdata-writer-v0.0.1"
}
```

2.3.6 Support Operations

Operations for Currency	
currency-register	Register new currency id
currency-policy-updater	Update currency policy
suffrage-infration	Increase amount of tokens

Operations for Account	
create-account	Create new account
key-updater	Update account keys
transfer	Transfer amount of tokens

Refer to *Seal Command* to check how to create those operations by commands.

2.4 Get Started

In this part, it will introduce how to run nodes on **Mitum Currency**.
For using Mitum Currency, you need to install *docker* and *golang* first.

If you are looking for the usage of **Mitum Blocksign**, visit mitum-data-blocksign.

2.4.1 About Mitum Currency Node

Mitum Blockchain network running Mitum Currency uses **PBFT-based ISAAC+ consensus protocol**.
In the *ISAAC+* consensus protocol, all nodes play the same role and participate in block generation.

Nodes participating in the network perform the following tasks.

- Making Proposal

- Block Verification
- Voting
- Storing Block
- Providing Digest API Service
- Transaction Requesting Collection

For more information on the **Mitum Blockchain** network, refer to Mitum Document.

2.4.2 Prerequisite

Database

Mitum Currency uses **MongoDB** as its main storage engine.

To run the Mitum currency node, you need to prepare mongodb first.

Installation and Setup

- Manual Installation Guide
- Using Docker Container,

```
$ docker run --name <db name> -it -p <host port>:<container port> -d mongo
```

- About DB setup, refer to *Configuration*.

Golang

Mitum Currency is developed using the programming language Go.

To create an executable binary, you need the source code to be built from.
We do not provide detailed instructions for installing the Go language here.
You must have the Golang installed with at least version 1.16 to build Mitum currency.

For more information, refer to How to Install Go.

2.4.3 Installation

- Please download the source code of Mitum Currency.
- Using **Git**,

```
$ git clone https://github.com/spikeekips/mitum-currency.git
```

- Build exe file.

```
$ cd mitum-currency

$ go build -ldflags="-X 'main.Version=v0.0.1-tutorial'" -o ./mc ./main.go

$ ./mc version
v0.0.1
```

To see all instructions of Mitum Currency, refer to *Command Line Interface*.

2.5 Configuration

The configuration for node setting is written in *YAML*.

2.5.1 address

Address of local node(alias for url address)

```
address: n0sas
```

2.5.2 genesis-operations

`genesis-operation` is a setting for the genesis operation that is executed when the network is initialized. `genesis-operation` contains the contents of the block that is initially created.

In the currency model, information on the main currency and genesis account must be set.

It registers the information about,

- Keys of the *genesis account* (key, weight, threshold)
- Initial balance
- Currency ID
- Fee policy of the currency to be created

For example,

```
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
          currency: MCC
          new-account-min-balance: "33"
    type: genesis-currencies
```

2.5.3 network

Specify the *domain address* or *IP address* of the node used in the network.
Address is needed to receive messages from node or client, and the communication process uses *quic* communication protocol.
Self-signed certificates can be used to set up test node. You can use it for only test and development nodes where security is not a big concern.

```
$ openssl genrsa -out mitum.key 4096

$ openssl req -x509 -new -nodes -key mitum.key -sha256 -days 1024 -out mitum.crt
```

For example,

```
network:
    bind: https://0.0.0.0:54321
    url: https://127.0.0.1:54321
    cert-key: mitum.key
    cert: mitum.crt
```

rate-limit

API interface of internet service allows connection to the client without restriction.
However, overflowing requests can ruin the performance of service.
To maintain the service stable, `rate limit` can be applied to the API service.

See Rate limiting.

Mitum supports *quic* based API service for communication within nodes, even none-suffrages.

Mitum Currency additionally supports *http2* based API service, called digest.

`rate-limit` applied to these API services.

```
network:
    bind: https://0.0.0.0:54321
    url: https://127.0.0.1:54321

    rate-limit:
        cache: "memory:?prefix=showme"
        preset:
            bad-nodes:
                new-seal: 3/2m
                blockdata: 4/m
        3.3.3.3:
            preset: bad-nodes
        4.4.4.4/24:
            preset: bad-nodes
            blockdata: 5/m
        127.0.0.1/24:
            preset: suffrage
```

- `cache`: cache for requests. At this time, supports "memory:" and "redis://<redis server>"
  - **memory**: memory cache
  - **redis://<redis server>**: cached in redis server
- `preset`: predefined rate limit settings.
  - For Mitum, `suffrage` and `world` presets are already defined. See launch/config/ratelimit.go in the source code.
  - You can make your own rate limit setting like bad-nodes.
- Rules:
  - **Rate-limit Settings** for a specific IP
  - Rules consist of *IP address* (or IP address range), `preset` and detailed `rate-limit` settings.
  - The IP address can be a single value or a range of IP addresses expressed in *CIDR* notation. * example : 3.3.3.3, 4.4.4.4/24, 127.0.0.1/24
  - Rate limit can be set through `preset` and additional `limits`.
  - `preset` can be pre-defined preset like `suffrage`, `world` or user-defined preset like `bad-nodes`.
  - Additional limit such as `blockdata:  5/m` can be added to the `preset`.
  - Rules will be checked by the defined order from upper to lower.
- Detailed limit:
  - The name of the API interface for Mitum, such as new-seal, used to set the limit can be found in RateLimitHandleMap (launch/config/ratelimit.go).
  - The name of the API interface for Mitum-currency can be found in RateLimitHandlerMap (digest/handler.go).

  - new-seal: 3/2m means new-seal interface allows 3 requests per 2 minutes to the specified IP or IP range.
  - See the manner of time duration.
- Without any rules, by default no rate limit.

A limit value less than zero means unlimited.

For example,

```
4.4.4.4/24:
preset: bad-nodes
blockdata: -1/m
```

The zero limit value means that the request is blocked.

For example,

```
4.4.4.4/24:
    preset: bad-nodes
    blockdata: 0/m
```

2.5.4 network-id

`network id` acts like an identifier that **identifies a network**.
All nodes on the same network have the same `network id` value.

For example,

```
network-id: mitum
```

2.5.5 keypair

Enter the **node's private key**.

For example,

```
privatekey: Kxt22aSeFzJiDQagrvfXPWbEbrTSPsRxbYm9BhNbNJTsrbPbFnPAmpr
```

See *Key Command* to learn how to create a key pair.

2.5.6 storage

Specify the *file system path* and *mongodb database address* of blockchain data storage.
If blockdata setting is missing, *blockdata* > *path* is set to a folder called *blockdata* in the current path by default.

For example,

```
storage:
blockdata:
    path: ./mc-blockfs
database:
    uri: mongodb://127.0.0.1:27017/mc
```

`port number` should be the same as that of when running docker.

2.5.7 suffrage

nodes

Set addresses for suffrage nodes participating in consensus.

The alias name of the local node is `n0sas`.
If `n0`, `n1`, `n2`, `n3` nodes are included in the suffrage nodes, it can be set as follows.

```
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas
```

If the `n0` node, which is a local node, is not included in the suffrage nodes, the local node becomes a *None-suffrage* node and serves only as a *syncing node*.

- The *Syncing node* does not participate in consensus and only syncs the generated block data.
- The *None-suffrage* node handles only the seal containing the operation.
- The *None-suffrage* node does not process ballots and proposals related to voting between nodes.
- When the *None-suffrage* node stores the operation seal, it broadcasts the seal to the suffrage nodes.

If the *None-suffrage* node does not add other nodes to the suffrage node, or does not configure other suffrage nodes, operation seal cannot be processed.

```
suffrage:
    nodes:
        - n1sas
        - n2sas
        - n3sas
```

2.5.8 sync-interval

None-suffrage node periodically syncs block data.

The default interval is 10 seconds.
You can change the interval value through the `sync-interval` setting.

```
sync-interval: 3s
```

2.5.9 nodes

Write the `address` (alias for the address), `public key`, and `url` (ip address) of known nodes in the blockchain network.

- If not written, it operates as a **standalone node**.
- If the node is a suffrage node and the node discovery function is used, the `url` of the node is not required.
- However, if the node is not a suffrage node, the ``url``s of the suffrage nodes must be included.

Mitum nodes use *CA signed certificate* (public certificate) by default.

- If certificate related settings are not made in *Network config*, the node uses *self-signed certificate*.
- If other Mitum nodes use self-signed certificate, `tls-insecure: true` should be set to all the nodes which use self-signed certificate.

```
(In case of suffrage node)

nodes:
    - address: n1sas
    publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
    tls-insecure: true
    - address: n2sas
    publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
    tls-insecure: true
    - address: n3sas
```

(continues on next page)

(continued from previous page)

```
    publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
    tls-insecure: true
```

```
(If it is not a suffrage node)

nodes:
    - address: n1sas
    publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
    url: https://127.0.0.1:54331
    tls-insecure: true
    - address: n2sas
    publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
    url: https://127.0.0.1:54341
    tls-insecure: true
    - address: n3sas
    publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
    url: https://127.0.0.1:54351
    tls-insecure: true
```

2.5.10 digest

Specify the *mongodb address* that stores the data to be provided by the *API* and the *IP address* of the API access.

```
digest:
    network:
        bind: https://localhost:54320
        url: https://localhost:54320
        cert-key: mitum.key
        cert: mitum.crt
```

2.5.11 tutorial.yml

This is an example of **standalone** node configuration.

```
address: mc-nodesas
privatekey: Kxt22aSeFzJiDQagrvfXPWbEbrTSPsRxbYm9BhNbNJTsrbPbFnPAmpr
storage:
    database:
        uri: mongodb://127.0.0.1:27017/mc
    blockdata:
        path: ./mc-blockfs
network-id: mitum
network:
    bind: https://0.0.0.0:54321
    url: https://127.0.0.1:54321
    cert-key: mitum.key
```

(continues on next page)

(continued from previous page)

```
    cert: mitum.crt
genesis-operations:
    - type: genesis-currencies
    account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
                weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
            currency: MCC
            new-account-min-balance: "33"
            feeer:
                type: fixed
                amount: 1
policy:
    threshold: 100
suffrage:
    nodes:
        - mc-nodesas

digest:
    network:
        bind: https://0.0.0.0:54320
        url: https://127.0.0.1:54320
        cert-key: mitum.key
        cert: mitum.crt
```

2.6 Run

Here we will explain the process for running the node.

Note:

- A node can find out the addresses of all nodes by using the node discovery protocol.
- *Digest API* is included in Mitum Currency, so API service is provided by default.
- Please check *Configuration* for Digest setting.
- If Digest is not set, data for API service must be processed separately.

2.6.1 Running the Standalone Node

Before running a node, please prepare tutorial.yml by refering to *Configuration*.

node init

First, the *genesis block* and *genesis account* must be created. The main currency is issued through the creation of the *genesis block* and stored in the balance of the *genesis account*.

- tutorial.yml : config file

```
$ ./mc node init --log-level info ./tutorial.yml
2021-06-10T05:13:09.232802Z INF dryrun? dryrun=false module=command-init
2021-06-10T05:13:09.235942Z INF prepare to run module=command-init
2021-06-10T05:13:09.236013Z INF prepared module=command-init
2021-06-10T05:13:09.780335419Z INF genesis block created block={"hash":
↪"6HjkXEhTNhPzUTG167jsTEany3dHebDQ5cKGNTNEzcgh","height":0} module=command-init
2021-06-10T05:13:10.786661419Z INF stopped module=command-init
...

$ echo $?
0
```

Note: If there is already a saved block data,, an error `environment already exists:  block=0` occurs. To reset the error and ignore it, run it by adding the `--force` option.

```
$ ./mc --log-level info init ./tutorial.yml --force
```

node run

When the node starts to run, the blockchain's storage status and consensus participation status are changed to *SYNC*, *JOIN*, and *CONSENSUS* modes, and block creation starts.

```
$ ./mc node run --log-level info ./tutorial.yml
2021-06-10T05:14:08.225487Z INF dryrun? dryrun=false module=command-run
2021-06-10T05:14:08.228797Z INF prepare to run module=command-run
2021-06-10T05:14:08.228869Z INF prepared module=command-run
2021-06-10T05:14:09.706271049Z INF new blocks found to digest last_block=-2 last_
↪manifest=0 module=command-run
2021-06-10T05:14:09.827980049Z INF digested new blocks module=command-run
2021-06-10T05:14:09.828967049Z INF trying to start http2 server for digest API␣
↪bind=https://localhost:54320 module=command-run publish=https://localhost:54320
2021-06-10T05:14:11.894638049Z INF new block stored block={"hash":
↪"CC57VpSKPozBRABPnznyMk6QY4GHn7CiSH4zSZBs8Rri","height":1,"round":0} elapsed=17.970959␣
↪module=basic-consensus-state proposal_
↪hash=DJBgmoAJ4ef7h7iF6E3gTQ83AjWxbGDGQrmDSiQMrfya voteproof_
↪id=BAg2HCNfBenFebuCM4P4HkDfF1off8FCBcSejdK1j7w6
2021-06-10T05:14:11.907600049Z INF block digested block=1 module=digester
```

If the node is a suffrage node, the addresses of other live suffrage nodes can be found using the *Node discovery protocol*. The node discovery feature is only supported when the node is a suffrage node.

- When the suffrage node starts up, it is possible to determine the network information of all suffrage nodes without publishing url information of all suffrage nodes.
- For node discovery, a node must set the address of one or more suffrage nodes it knows to a discovery url at startup.

To specify the discovery url, use the `-discovery` command line option.

```
$ ./mc node run n0.yml --discovery "https://n1#insecure" --discovery "https://n2#insecure
↪"
```

- Even if a node does not set the discovery url by itself, if another suffrage node designates this node as a discovery node, the publish url of other nodes is known by the gossip protocol. If the nodes specified by discovery are not running, it keeps trying until it succeeds.
- Again, node discovery only works with suffrage nodes. For nodes not included in the suffrage node list, the urls of other suffrage nodes are still specified in the node settings.
- If you set the log level to info, you can easily check the information of the newly created block.

`-log` command line option can collect logs to the specific files.

Mitum dumps huge debugging log messages, including *quic* (http) request message like this,

```
"l":"debug","module":"http2-server","ip":"127.0.0.1","user_agent":"Mozilla/5.0␣
↪(Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/
↪14.0.3 Safari/605.1.15","req_id":"c30q3kqciaejf9nj79c0","status":200,"size":2038,
↪"duration":0.541625,"content-length":0,"content-type":"","headers":{"Accept-Language":[
↪"en-us"],"Connection":["keep-alive"],"Upgrade-Insecure-Requests":["1"]},"host":"127.0.
↪0.1:54320","method":"GET","proto":"HTTP/1.1","remote":"127.0.0.1:55617","url":"/","t":
↪"2021-06-10T05:23:31.030086621Z","caller":"/Users/soonkukkang/go/pkg/mod/github.com/
↪spikeekips/mitum@v0.0.0-20210609043008-298f37780037/network/http.go:61","m":"request"
```

`-network-log` command line option can collect these request messages to the specific files.

```
$ ./mc node run \
    --log-level debug \
    --log-format json \
    --log ./mitum.log \
    --network-log ./mitum-request.log \
    ./tutorial.yml
```

Multiple file can be set to `-network-log` and `-log`.

In Mitum Currency, `-network-log` option will also collect the requests log from *digest API* (http2). `-network-log` option is only available in `node run` command.

Lookup Genesis Account

You can check *genesis account* information through block files saved in the file system.

For example,

```
$ find blockfs -name "*-states-*" -print | xargs -n 1 gzcat | grep '^{' | jq '. |␣
↪select(.key == "9g4BAB8nZdzWmrsAomwdvNJU2hA2psvkfTQ5XdLn4F4r-mca:account") | [
↪"height: "+(.height|tostring), "state_key: " + .key, "address: " + .value.value.
↪address, .operations, .value.value.keys.keys, .value.value.keys.threshold]'
[
    "height: 0",
    "state_key: 9g4BAB8nZdzWmrsAomwdvNJU2hA2psvkfTQ5XdLn4F4r-mca:account",
    "address: CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
    [
        "ECSDvWwxcjbEw2F3E6n6pyQXMsZn2uy7msX19XXDCYi8"
    ],
    [
        {
        "_hint": "mitum-currency-key-v0.0.1",
        "weight": 100,
        "key": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu"
        }
    ],
    100
]
```

```
$ find blockfs -name "*-states-*" -print | xargs -n 1 gzcat | grep '^{' |jq '. | select(.
↪key == "9g4BAB8nZdzWmrsAomwdvNJU2hA2psvkfTQ5XdLn4F4r-mca-MCC:balance") | [ "height:
↪"+(.height|tostring), "state_key: " + .key, "balance:" + .value.value.amount]'
[
    "height: 0",
    "state_key: 9g4BAB8nZdzWmrsAomwdvNJU2hA2psvkfTQ5XdLn4F4r-mca-MCC:balance",
    "balance:99999999999999999999"
]
```

- *height*, *address* of genesis account at `0`, `CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca` is saved in block.

Lookup using Digest API

Account information can also be checked through *Digest API*.

The api address according to the digest setting *Configuration* is https://localhost:54320.

Check genesis account through account information.

```
$ curl --insecure http://localhost:54320/account/
↪CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca | jq '{_embedded}'
{
    "_embedded": {
        "_hint": "mitum-currency-account-value-v0.0.1",
        "hash": "6vCuuiqaYtNGfPbqfDqA234kiDoueWejd7jMs7dwvq5U",
        "address": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
        "keys": {
            "_hint": "mitum-currency-keys-v0.0.1",
            "hash": "9g4BAB8nZdzWmrsAomwdvNJU2hA2psvkfTQ5XdLn4F4r",
            "keys": [
                {
                "_hint": "mitum-currency-key-v0.0.1",
                "weight": 100,
                "key": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu"
                }
            ],
            "threshold": 100
        },
        "balance": [
            {
                "_hint": "mitum-currency-amount-v0.0.1",
                "amount": "99999999999999999999",
                "currency": "MCC"
            }
        ],
        "height": 0,
        "previous_height": -2
    }
}
```

2.6.2 Build Multi Nodes Network

Order of Execution

1. When executing a multi node, the first node that creates the *genesis block* must be determined. The first node creates the *genesis block* through the `node init` command. Nodes other than the one that creates the *genesis block* do not need to execute the `init` command.
2. The first node executes the node through the `run` command after `init`.
3. Other nodes also execute each node through the `run` command.

4. Other nodes follow the block of the first node through the *sync* process, and the nodes create blocks through the *consensus* process.

If there are 4 nodes and n0 node is the first node, the execution order is as follows. If all four nodes are suffrage nodes, nodes must set at least one other node *publish url* as the *discovery url* for node discovery.

```
# n0 node
$ ./mc node init --log-level info ./n0.yml
$ ./mc node run --log-level info ./n0.yml --discovery "https://n1#insecure"
```

```
# n1 node
$ ./mc node run --log-level info ./n1.yml --discovery "https://n0#insecure"
```

```
# n2 node
$ ./mc node run --log-level info ./n2.yml --discovery "https://n0#insecure"
```

```
# n3 node
$ ./mc node run --log-level info ./n3.yml --discovery "https://n0#insecure"
```

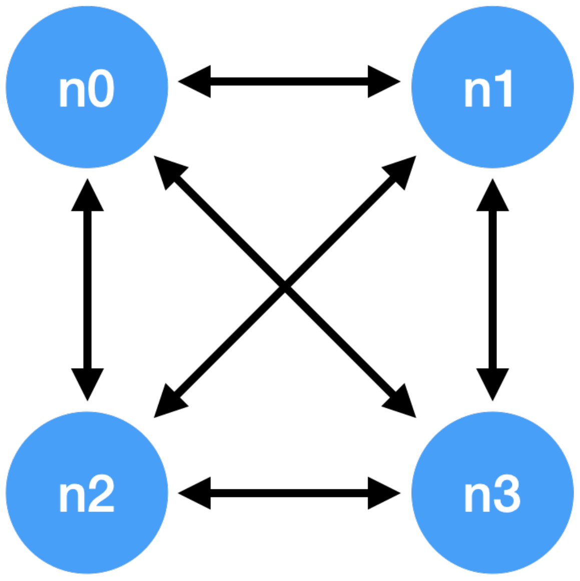
Note: If running in the same network, nodes should have the same value for the next item in the configuration file.

- `genesis-operations`
- `network-id`

Four Suffrage Nodes

Let's suppose we are in case of operating suffrage 4 nodes.

First, prepare **a separate yml configuration file for each node**.
`n0`, `n1`, `n2`, `n3` are all suffrage nodes.



Depending on the configuration of the node, it is necessary to configure the nodes participating in consensus.

```
# Only ``suffrage`` and ``nodes`` part of configuration of suffrage nodes

suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas

nodes:
    - address: n0sas
    publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
    tls-insecure: true
    - address: n1sas
    publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
    tls-insecure: true
    - address: n2sas
    publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
    tls-insecure: true
    - address: n3sas
    publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
    tls-insecure: true
```

The following one is an example of the full yml configuration for all nodes.

```
# n0 node
```

(continues on next page)

(continued from previous page)

```
address: n0sas
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54321
    url: https://127.0.0.1:54321
network-id: mitum
policy:
    threshold: 100
privatekey: Kxt22aSeFzJiDQagrvfXPWbEbrTSPsRxbYm9BhNbNJTsrbPbFnPAmpr
publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
storage:
    blockdata:
        path: ./n0_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27017/n0_mc
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas
nodes:
    - address: n1sas
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      tls-insecure: true
    - address: n2sas
      publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      tls-insecure: true
```

```
# n1 node
address: n1sas
genesis-operations:
    - account-keys:
        keys:
            - privatekey: L5GTSKkRs9NPsXwYgACZdodNUJqCAWjz2BccuR4cAgxJumEZWjokmpr
              publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
```

(continues on next page)

(continued from previous page)

```
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54331
    url: https://127.0.0.1:54331
network-id: mitum
policy:
    threshold: 100
privatekey: L4R2AZVmxWUiF2FrNEFi6rHwCTdDLQ1JuQHji69SbMcmWUdNMUSFmpr
publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
storage:
    blockdata:
        path: ./n1_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27018/n1_mc
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas
nodes:
    - address: n0sas
      publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
      tls-insecure: true
    - address: n2sas
      publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      tls-insecure: true
```

```
# n2 node
address: n2sas
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54332
    url: https://127.0.0.1:54332
network-id: mitum
policy:
    threshold: 100
privatekey: L3Szj4t3w33YLsGFGeaB3v1vwae82yp5KWPcT7v1Y4WyQkAH7eCRmpr
publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
```

(continues on next page)

(continued from previous page)

```
storage:
    blockdata:
        path: ./n2_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27019/n2_mc
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas
nodes:
    - address: n0sas
      publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
      tls-insecure: true
    - address: n1sas
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      tls-insecure: true
```

```
# n3 node
address: n3sas
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54333
    url: https://127.0.0.1:54333
network-id: mitum
policy:
    threshold: 100
privatekey: KwxfBSzwevSggJz2grf8FWrjvXzrctY3WismTy6GNdJpWXe5tF5Lmpr
publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
storage:
    blockdata:
        path: ./n3_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27020/n3_mc
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
```

(continues on next page)

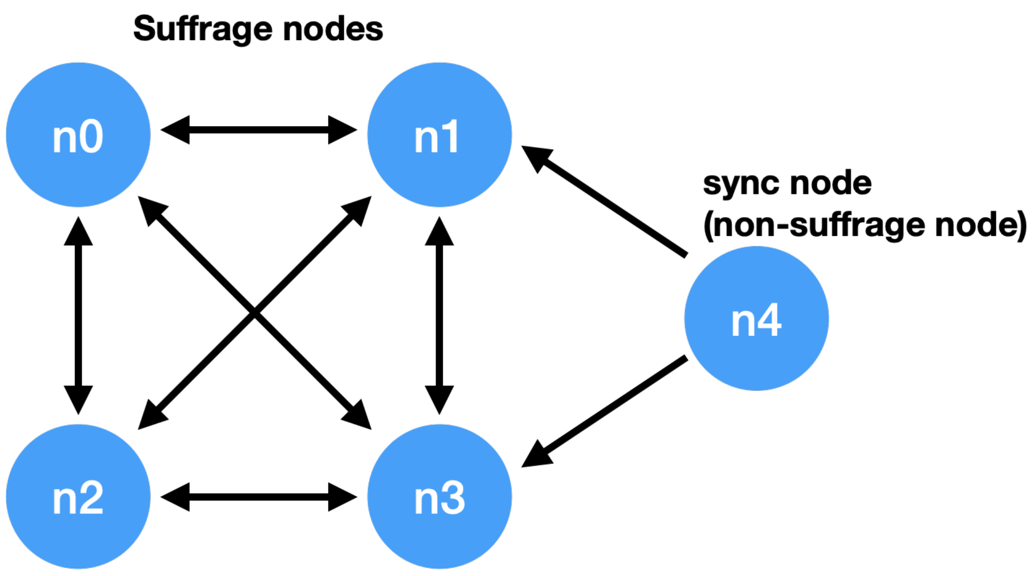
(continued from previous page)

```
        - n3sas
nodes:
    - address: n0sas
      publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
      tls-insecure: true
    - address: n1sas
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      tls-insecure: true
    - address: n2sas
      publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
      tls-insecure: true
```

Four Suffrage Nodes and One Sync Node

In case of operating four suffrage nodes and one sync node(non-suffrage node),

Prepare a separate yml configuration file for each node.
`n0`, `n1`, `n2`, `n3` are suffrage nodes and `n4` is the sync node.



Only `suffrage` and `nodes` part of configuration of suffrage nodes(n0, n1, n2, n3) are like,

```
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas
```

(continues on next page)

(continued from previous page)

```
nodes:
    - address: n0sas
      publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
      tls-insecure: true
    - address: n1sas
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      tls-insecure: true
    - address: n2sas
      publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      tls-insecure: true
```

Only `suffrage` and `nodes` part of configuration of sync node(n4) are like,

```
# suffrage and nodes part of configuration

suffrage:
    nodes:
        - n1sas
        - n3sas

nodes:
    - address: n1sas
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      url: https://127.0.0.1:54331
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      url: https://127.0.0.1:54351
      tls-insecure: true
```

The following one is an example of the full yml configuration for all nodes.

```
# n0 node(Suffrage node)

address: n0sas
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
```

(continues on next page)

(continued from previous page)

```
        - balance: "99999999999999999999"
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54321
    url: https://127.0.0.1:54321
network-id: mitum
policy:
    threshold: 100
privatekey: Kxt22aSeFzJiDQagrvfXPWbEbrTSPsRxbYm9BhNbNJTsrbPbFnPAmpr
publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
storage:
    blockdata:
        path: ./n0_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27017/n0_mc
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas
nodes:
    - address: n1sas
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      tls-insecure: true
    - address: n2sas
      publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      tls-insecure: true
```

```
# n1 node(Suffrage node)

address: n1sas
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54331
    url: https://127.0.0.1:54331
network-id: mitum
policy:
    threshold: 100
```

(continues on next page)

(continued from previous page)

```
privatekey: L4R2AZVmxWUiF2FrNEFi6rHwCTdDLQ1JuQHji69SbMcmWUdNMUSFmpr
publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
storage:
    blockdata:
        path: ./n1_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27018/n1_mc
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas
nodes:
    - address: n0sas
      publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
      tls-insecure: true
    - address: n2sas
      publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      tls-insecure: true
```

```
# n2 node(Suffrage node)

address: n2sas
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54332
    url: https://127.0.0.1:54332
network-id: mitum
policy:
    threshold: 100
privatekey: L3Szj4t3w33YLsGFGeaB3v1vwae82yp5KWPcT7v1Y4WyQkAH7eCRmpr
publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
storage:
    blockdata:
        path: ./n2_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27019/n2_mc
suffrage:
    nodes:
```

(continues on next page)

(continued from previous page)

```
        - n0sas
        - n1sas
        - n2sas
        - n3sas
nodes:
    - address: n0sas
      publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
      tls-insecure: true
    - address: n1sas
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      tls-insecure: true
```

```
# n3 node(Suffrage node)

address: n3sas
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54333
    url: https://127.0.0.1:54333
network-id: mitum
policy:
    threshold: 100
privatekey: KwxfBSzwevSggJz2grf8FWrjvXzrctY3WismTy6GNdJpWXe5tF5Lmpr
publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
storage:
    blockdata:
        path: ./n3_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27020/n3_mc
suffrage:
    nodes:
        - n0sas
        - n1sas
        - n2sas
        - n3sas
nodes:
    - address: n0sas
      publickey: skRdC6GGufQ5YLwEipjtdaL2Zsgkxo3YCjp1B6w5V4bDmpu
      tls-insecure: true
    - address: n1sas
```

(continues on next page)

(continued from previous page)

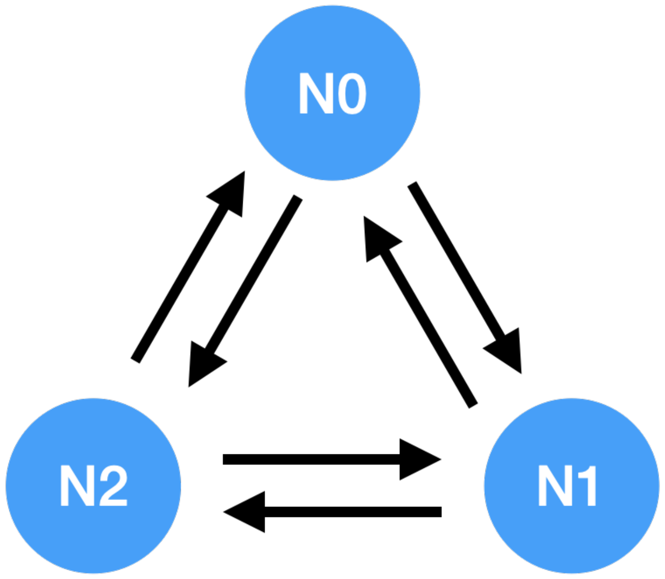
```
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      tls-insecure: true
    - address: n2sas
      publickey: wfVsNvKaGbzB18hwix9L3CEyk5VM8GaogdRT4fD3Z6Zdmpu
      tls-insecure: true
```

```
# n4 node(Sync node)

address: n4sas
genesis-operations:
    - account-keys:
        keys:
            - publickey: rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu
              weight: 100
        threshold: 100
    currencies:
        - balance: "99999999999999999999"
          currency: MCC
    type: genesis-currencies
network:
    bind: https://0.0.0.0:54334
    url: https://127.0.0.1:54334
network-id: mitum
policy:
    threshold: 67
privatekey: KyKM3JtH8M9iBQrcFx4Lubi13Bg8pUPVYvxhihEfkiiqRRWYjjr4mpr
publickey: 2BQkVjJpMdx4BFEhfTtf1oTaG4nLN148Dfax3ZnWybA2bmpu
storage:
    blockdata:
        path: ./n4_data/blockfs
    database:
        uri: mongodb://127.0.0.1:27021/n4_mc
suffrage:
    nodes:
        - n1sas
        - n3sas
nodes:
    - address: n1sas
      publickey: ktJ4Lb6VcmjrbexhDdJBMnXPXfpGWnNijacdxD2SbvRMmpu
      url: https://127.0.0.1:54331
      tls-insecure: true
    - address: n3sas
      publickey: vAydAnFCHoYV6VDUhgToWaiVEtn5V4SXEFpSJVcTtRxbmpu
      url: https://127.0.0.1:54333
      tls-insecure: true
```

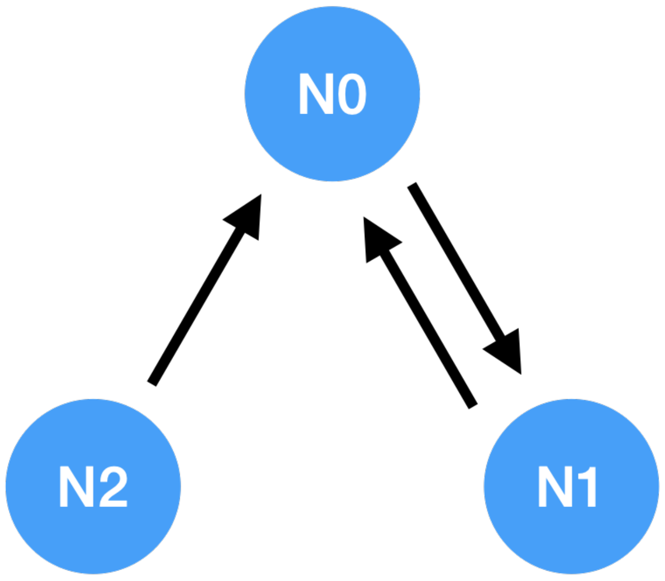
Node Discovery Scenario

This is an example of a scenario for *Node Discovery*.

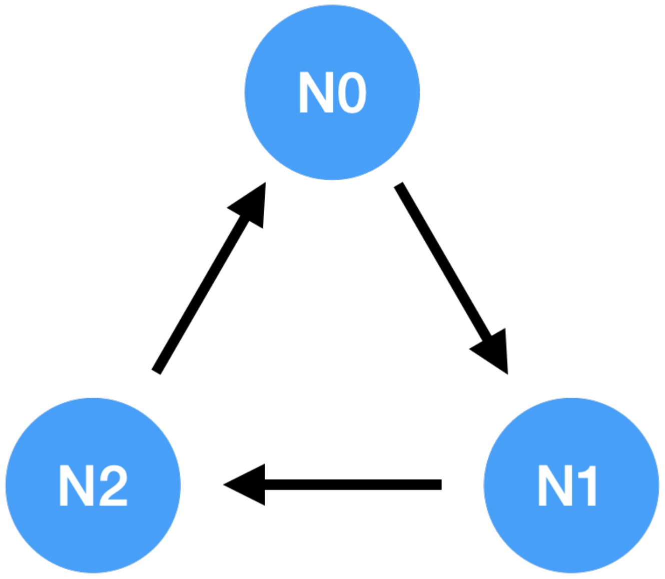
```
case 0

All nodes are looking at each other
discoveries of n0: n1, n2
discoveries of n1: n0, n2
discoveries of n2: n0, n1
all joined
```



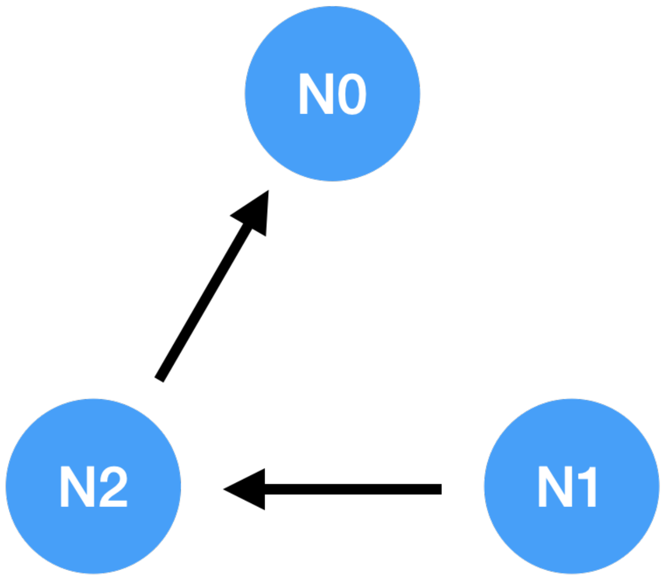
```
case 1

All nodes are looking at the same node and only one node is looking at the other node.
discoveries of n0: n1
discoveries of n1: n0
discoveries of n2: n0
all joined
```



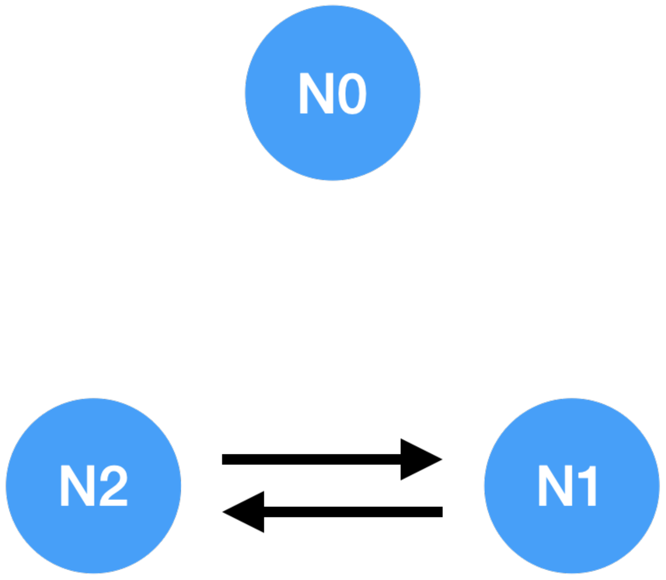
```
case 2

All nodes are looking at each other.
discoveries of n0: n1
discoveries of n1: n2
discoveries of n2: n1
all joined
```



```
case 3

One node is looking at no one, but another node is looking at it.
discoveries of n0: none
discoveries of n1: n2
discoveries of n2: n0
all joined
```



```
case 4

A node sees no one, but no other nodes see it.
discoveries of n0: none
discoveries of n1: n2
discoveries of n2: n1
n1, n2: connected to each other
n0: disconnected
```



2.7 Node Handover

Here we will explain the process of **node handover**.

2.7.1 What is Handover?

Handover is a feature that allows **a running node to be replaced without stopping**.

- Nodes participating in the consensus process in the entire network may need to be replaced due to various circumstances.
- There may be cases where the program running the node needs to be updated or moved to another cloud service, or the node fails. In such cases, you can replace an already running node with an alternate node without stopping it.

Handover is one of the key features that can support public networks in the future.

2.7.2 Handover Scenario

- **Node A**: the running node
- **Node A-sub**: the node to be replaced
- Condition: *A-sub* must have the same configuration as *A*. However, the value of `network.url` must be different.

Node *A*'s Configuration

```
address: Asas
network-id: mitum
network:
    url: https://172.17.0.1:54321
privatekey: KyfqCLSEyfUhskZ63WtVH3m3pGgnurFHuTgkgu73Pgyjf8sxbp8Rmpr
```

Node *A-sub*'s Configuration

```
address: Asas
network-id: mitum
network:
    url: https://172.17.0.2:54321
privatekey: KyfqCLSEyfUhskZ63WtVH3m3pGgnurFHuTgkgu73Pgyjf8sxbp8Rmpr
```

2.7.3 How to Run

Under the scenario above, it follows the steps below.

- *A* is running as a suffrage node and wants to replace it with *A-sub*.
- After confirming that *A* is in **CONSENSUS** state, run *A-sub* with `node run` command using the above config.
- *A-sub* performs synchronization by collecting previous block data and enters **SYNCING** state.
- After confirming that the *A-sub* has changed to **SYNCING** state, execute the `start-handover` command on *A-sub* node.

```
$ mitum-currency node start-handover \
    "Asas" \
    "KyfqCLSEyfUhskZ63WtVH3m3pGgnurFHuTgkgu73Pgyjf8sxbp8Rmpr" \
    "mitum" \
    "https://172.17.0.2:54321"
```

- When *B* finishes syncing, *Handover* starts.
- *A* is switched from **CONSENSUS** state to **SYNCING** state, and *A-sub* performs the consensus process on behalf of *A*.
- Other nodes in the suffrage network add *A-sub* as a suffrage node instead of *A* and proceed with the consensus process.
- Operation Seals delivered to *A* are delivered to other nodes.

What If a start-handover is sent to A after the Handover is over?

- *A* replaced by *A-sub* is converted to **SYNCING** state.
- After the handover is finished, *A* is internally converted to the same state as *A-sub* before the `start-handover` command.
- If a `start-handover` command is delivered to *A* in this state, from then on, *A* attempts to replace *A-sub*.

How can I check that the start-handover is finished?

- When checking *NodeInfo* of *A-sub*, it is changed to **CONSENSUS** state.
- Once *A*’s *NodeInfo* is checked, if it is changed to **SYNCING** state, handover is succesfully completed.

2.8 Command Line Interface

In this part, we will introduce the commands supported by Mitum Currency and how to utilize them.

There are seven major commands it supports,

- `version`
- `node`
- `key`

- seal
- storage
- deploy
- quic-client

You may be familiar with node command if you have already been to Run page.

It is easy to use the version and quic-client commands. They are explained in Others. We will now explain the rest of the commands one by one.

2.8.1 Summary

These are all the commands that Mitum Currency provides.

```
$ ./mc --help

Usage: mitum-currency <command>

mitum-currency tool

Flags:
    -h, --help    Show context-sensitive help.

Commands:
    version                         version

    node                                node
        init                            initialize node
            <node design file>          node design file
        run                             run node
            <node design file>          node design file
        info                            node information
            <node url>                  remote mitum url
        start-handover                  start handover
            <node address>
            <private key of node>
            <network-id>
            <new node url>              new node url

    key                                 key
        new                             new keypair
        verify                          verify key
            <key>                       key
        address                         generate address from key
            [<threshold>]               threshold for keys (default: 100)
            [<key> ...]                 key for address (ex: "<public key>,<weight>")
        sign                            signature signing
```

(continues on next page)

(continued from previous page)

```
        <privatekey>                privatekey
        <signature base>            signature base for signing

  seal                              seal
    send                            send seal to remote mitum node
        <privatekey>                privatekey for sign
    create-account                  create new account
        <privatekey>                privatekey to sign operation
        <sender>                    sender address
        <currency-amount> ...       amount (ex: "<currency>,<amount>")
    transfer                        transfer big
        <privatekey>                privatekey to sign operation
        <sender>                    sender address
        <receiver>                  receiver address
        <currency-amount> ...       amount (ex: "<currency>,<amount>")
    key-updater                     update keys
        <privatekey>                privatekey to sign operation
        <target>                    target address
        <currency>                  currency id
    currency-register               register new currency
        <privatekey>                privatekey to sign operation
        <currency-id>               currency id
        <genesis-amount>            genesis amount
        <genesis-account>           genesis-account address for genesis balance
    currency-policy-updater         update currency policy
        <privatekey>                privatekey to sign operation
        <currency-id>               currency id
    suffrage-inflation              suffrage inflation operation
        <privatekey>                privatekey to sign operation
        <inflation item> ...        ex: "<receiver address>,<currency>,<amount>"
    sign                            sign seal
        <privatekey>                sender's privatekey
    sign-fact                       sign facts of operation seal
        <privatekey>                sender's privatekey

  storage                           storage
    download                        download block data
        <data type>                 data type of block data,
                                    {manifest,operations,operations_tree,states,
↪states_tree,init_voteproof,accept_voteproof,suffrage_info,proposal all}
        <height> ...                heights of block
    verify-blockdata                verify block data
        <blockdata path>
    verify-database                 verify database
        <database uri>
        <blockdata path>
    clean                           clean storage
        <node design file>          node design file
    clean-by-height                 clean storage by height
        <node design file>          node design file
        <height>                    height of block
    restore                         restore blocks from blockdata
```

(continues on next page)

(continued from previous page)

```
            <node design file>          node design file
        set-blockdatamaps               set blockdatamaps
            <deploy key>
            <maps file>                 set blockdatamap file
            [<node url>]                remote mitum url; default: quic://localhost:54321

    deploy                              deploy
        key                             deploy key
            new                         request new deploy key
                <private key of node>
                <network-id>
                [<node url>]            remote mitum url; default: quic://localhost:54321
        keys                            deploy keys
            <private key of node>
            <network-id>
            [<node url>]                remote mitum url; default: quic://localhost:54321
        key                             deploy key
            <deploy key>
            <private key of node>
            <network-id>
            [<node url>]                remote mitum url; default: quic://localhost:54321
        revoke                          revoke deploy key
            <deploy key>
            <private key of node>
            <network-id>
            [<node url>]                remote mitum url; default: quic://localhost:54321

    quic-client                         quic-client
        <node url>                      remote mitum url

Run "mitum-currency <command> --help" for more information on a command.
```

2.9 Node Command

The node command initializes nodes and runs nodes.

The subcommands of the node command are as follows.

- init
- run
- start-handover
- info

2.9.1 init

The `init` command is used for **initializing the node** with the node design file containing the node configuration.

See *node init* for a detailed explanation of the `init` command.

```
$ ./mc node init <node design file>
```

2.9.2 run

The `run` command is used for **running the node** with the node design file containing the node configuration.

See *node run* for a detailed explanation of the `run` command.

```
$ ./mc node run <node design file>
```

2.9.3 start-handover

The `start-handover` command is used for **replacing the running node** with another node.

See *Node Handover* for a detailed explanation of the `start-handover` command.

```
$ ./mc node start-handover <node address> <private key of node> <network-id> <new node␣
↪url>
```

2.9.4 info

The `info` command is used for **getting the information of the remote node** with the node's url.

```
$ ./mc node info <node url>
```

EXAMPLE

```
$ ./mc node info https://127.0.0.1:54321 --tls-insecure | jq
{
    "_hint": "mitum-currency-node-info-v0.0.1",
    "node": {
        "_hint": "base-node-v0.0.1",
```

(continues on next page)

(continued from previous page)

```
        "address": "mc-nodesas",
        "publickey": "27P4S2FdDALmg4QzShCDTDne1pe8y1H2bE2uQCVpnqWpumpu",
        "url": "https://127.0.0.1:54321"
    },
    "network_id": "bWl0dW0=",
    "state": "CONSENSUS",
    "last_block": {
        "_hint": "block-manifest-v0.0.1",
        "hash": "5Z2SFA6DqYg8KdRPAD4uXAM7wpPE6vjyQ5iWqu4sc1yP",
        "height": 421,
        "round": 0,
        "proposal": "3H5wmRqvnburtEMqvkLh11vetbbdsdvHAkJRM6L6nu3Z",
        "previous_block": "J3if3xYD1wUQxUnm52UpddHT4Dipsd35bYGQxurMGnXm",
        "block_operations": null,
        "block_states": null,
        "confirmed_at": "2021-06-10T07:04:31.378699784Z",
        "created_at": "2021-06-10T07:04:31.390856784Z"
    },
    "version": "v0.0.0",
    "url": "https://127.0.0.1:54321",
    "policy": {
        "network_connection_timeout": 3000000000,
        "max_operations_in_seal": 10,
        "max_operations_in_proposal": 100,
        "interval_broadcasting_init_ballot": 1000000000,
        "wait_broadcasting_accept_ballot": 1000000000,
        "threshold": 100,
        "interval_broadcasting_accept_ballot": 1000000000,
        "timeout_waiting_proposal": 5000000000,
        "timespan_valid_ballot": 60000000000,
        "interval_broadcasting_proposal": 1000000000,
        "suffrage": "{\"type\":\"\",\"cache_size\":10,\"number_of_acting\":1}"
    },
    "suffrage": [
        {
            "_hint": "base-node-v0.0.1",
            "address": "mc-nodesas",
            "publickey": "27P4S2FdDALmg4QzShCDTDne1pe8y1H2bE2uQCVpnqWpumpu",
            "url": "https://127.0.0.1:54321"
        }
    ]
}
```

2.10 Key Command

The `key` command creates keypairs, gets addresses from keys, and gets signature.

The subcommands of the `key` command are as follows.

- `new`
- `address`
- `sign`

Note: Keypair

- *Private key* and *public key* are created through keypair generation.
- The generated keypair is used to *create an account*, *register a keypair* of a node, and *create a signature* of operation and seal.

2.10.1 new

The `new` is used for **creating a new keypair**.

Random Keypair

Use the following to create a random keypair without any seed.

```
$ ./mc key new
```

EXAMPLE

```
$ ./mc key new
      hint: mpr
privatekey: L1ZERchoY53vC5TJQ3WnZEWmg97L2Utw5rgFrCwM7ekTu9zJkZYjmpr
 publickey: 28nFxuC5ETygieSGEYTkewwnCZseB4TNYGMRtxz31bvxzmpu
```

Keypair from Seed

Use the following to create a keypair from a seed. Note that the length of string seed must be longer than or equal to 36.

```
$ ./mc key new --seed <string seed>
```

EXAMPLE

```
$ ./mc key new --seed abcdefghijklmnopqrstuvwxyzABCDEFGHIJ
      hint: mpr
privatekey: KypAAGYtVFdTFLS8muPJhwfJBFCFHKSe594yYmKK3FPteh7sie4Dmpr
 publickey: 25BcZrcyiE3TD2BZEqkdDuaYB9zHxpdW82BNn8HkCLTijmpu
```

2.10.2 address

The `address` command is used for generating addresses from keys.

You should prepare (public key, weight) pairs and threshold for the account. Refer to below *Multi Sig Account* for details.

```
$ ./mc key address <threshold> [<publickey>,<weight>]
```

EXAMPLE

For example, let's say that the information of the account is given by the table below, then the `address` command is used as follows.

threshold	100
keys	{key: 21Sn1o…, weight: 50}, {key: utzCef…, weight: 50}

```
$ ./mc key address 100 21Sn1owHXRx336aaerU1WbbKjiZXMcrJsnxBHP9etNx6zmpu,50␣
↪utzCefA1Szmmt3rAwqW5yEhxK1x3hG3Y3yThEK3gZmv3mpu,50
37x8YoAGA93B3HmDVNterRf1NTgz9tfN1gQn4jYuBYCHmca
```

However, you won't get correct address if the keys of the account have been updated by the key-updater command. Refer to *key-updater*.

Multi Sig Account

- *Account* is a data structure that has *currency* and *balance* in Mitum Currency.
- *Account* has a unique value called *address* and can be identified through this.
- Register a public key for user's *Account authentication*.
- Mitum Currency accounts can register *multiple public keys* because **multi signatures are possible**.

For example, an account under following condition is available.

address	HjyXhhuHAZBGaEw2S5cKZhDwqVc1StbkJMtdgGm3F1dnmca
threshold	100
keys	{key: rd89Gx…, weight: 50}, {key: skRdC6…, weight: 50}
balance	{currency: MCC, amount: 10000}, {currency: MCC2, amount: 20000}

Note: There are several conditions that each account should follow.

- The range of `threshold` should be 1 <= threshold <= 100.
- The range of each `weight` should be 1 <= weight <= 100.
- The sum of every weight of the account should be greater than or equal to `threshold`.
- Each key must be a BTC compressed public key with suffix `mpu`.
- `mca` follows the address as a suffix.

These are examples of available account states.

CASE1 (single)

- threshold: 100
- keys: {key: rd89Gx…, weight: 100}

CASE2 (single)

- threshold: 50
- keys: {key: rd89Gx…, weight: 60}

CASE3 (multi)

- threshold: 100
- keys: {key: rd89Gx…, weight: 40}, {key: skRdC6…, weight: 30}, {key: mymMwq…, weight: 30}

CASE4 (multi)

- threshold: 50
- keys: {key: rd89Gx…, weight: 20}, {key: skRdC6…, weight: 20}, {key: mymMwq…, weight: 10}

Even in the same publickey combination, address will have different values if `weight` or `threshold` are different.

2.10.3 sign

The `sign` command is used for getting the signature of the private key for a specific message.

```
$ ./mc key sign <privatekey> <signature base>
```

EXAMPLE

```
$./mc key sign L5nDx2QtZVBPtJvUQ13cj3bMhC487JdxrwXTdS6JgzTvnSHestCxmpr bWVzc2FnZQ=
381yXZHrm73kGD8z7FAksBjxy49wPRWn3WRdP22befdbFff6WYSdK8rz9TLpFWuEW7rmmphF3rHkrvTPvhVQ5kXNGLmELBwZ
```

Note that signature base is string type encoded by *base64*.

2.11 Seal Command

The `seal` command helps execute various operations contained in the seal.

The subcommands related to **operation** are as follows.

- `create-account`
- `transfer`
- `key-updater`
- `currency-register`
- `currency-policy-updater`
- `suffrage-inflation`

The `seal` command also helps create a signature and sends a seal.

The subcommands related to **signature generation** and **transmission** are as follows.

- `send`
- `sign`
- `sign-fact`

Whether the operation has been successfully processed or not can be checked through the api.

For more information, please refer to *Confirming the Success of the Operation*.

2.11.1 create-account

The `create-account` command is used for creating an account.

```
$ ./mc seal create-account --network-id=NETWORK-ID-FLAG <privatekey> <sender> <currency,
↪amount> --key=KEY@... --threshold=
```

- KEY: <pub key, weight>

EXAMPLE

We will proceed with the process of creating two accounts, `ac0` and `ac1` as an example.

For how to create a keypair, please refer to *Key Command*.

The operation that creates account `ac0` is as follows.

1. **Create Single Sig Account**

We will create the account according to the following account information.

```
sender's account - who create new account
private key: L5GTSKkRs9NPsXwYgACZdodNUJqCAWjz2BccuR4cAgxJumEZWjokmpr
address: Gu5xHjhos5WkjGo9jKmYMY7dwWWzbEGdQCs11QkyAhh8mca

ac0
public key - weight: cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu - 100
threshold: 100
initial balance: 500 MCC
```

```
$ NETWORK_ID="mitum"

$ SENDER_PRV=L5GTSKkRs9NPsXwYgACZdodNUJqCAWjz2BccuR4cAgxJumEZWjokmpr

$ SENDER_ADDR=Gu5xHjhos5WkjGo9jKmYMY7dwWWzbEGdQCs11QkyAhh8mca

$ AC0_PUB=cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu

$ ./mc seal create-account --network-id=$NETWORK_ID $SENDER_PRV $SENDER_ADDR MCC,500 --
↪key=$AC0_PUB,100 --threshold=100 | jq
{
    "_hint": "seal-v0.0.1",
    "hash": "Xr7HS7rnbfxTrNbr6qRJ64on6KFuMzvJf5Z6BGqVZsX",
    "body_hash": "EJ93htxhUh2edJhBujMCHhpvGGHQoBic8KQ7VzggxKw1",
    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
```

(continues on next page)

(continued from previous page)

```
    "signature":
↪"381yXZUffVp3gmKD2WJA6756SeDy16d3PF6Ym15HBL89rs1YhT1cW4zVnWD17mhBdhfhutu3848GPd9zTMDqUFmkE8rUWmCs
↪",
    "signed_at": "2021-06-10T14:06:17.60152Z",
    "operations": [
        {
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "8ezjZDuC44U2ZFPDkebMyLEYNQBPUUnRjHyfSTeQs9gk",
            "fact": {
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "F1o51xXWnnQYUVV6JA44beJeKKxuJi3Tv8DzvREodHhA",
                "token": "MjAyMS0wNi0xMFQxNDowNjoxNy41OTczMDNa",
                "sender": "Gu5xHjhos5WkjGo9jKmYMY7dwWWzbEGdQCs11QkyAhh8mca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLu",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "500",
                                "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
                    "signature":
↪"381yXYyRo91cqu5gFp5GtHWCiYmsssbFxx95MaL8gH4koBCZ5AfnRqYEpWMxcxgKmeEWsRPVJ8zWytAMLiA9zQes9qGnbcj8
↪",
                    "signed_at": "2021-06-10T14:06:17.601089Z"
                }
            ],
            "memo": ""
        }
    ]
```

(continues on next page)

(continued from previous page)

```
}
```

The above json messages are put in the seal and sent to the node.

2. **Create Multi Sig Account**

Note:

- In Mitum Currency, two or more operations signed by one account cannot be processed in one block.
- For example, two respective operations that send 5 amount from `ac0` to `ac1` and `ac2` cannot be processed at the same time.
- In this case, only the operation that arrived first is processed and the rest are ignored.

Suppose that the sender is trying to create `ac0` and `ac1` at the same time using only one seal. Then the sender should include items for both `ac0` and `ac1`.

This means that for the operation to be processed successfully, the sender should create and send only one operation that creates two accounts in the seal. **Do not make multiple separate operations with the same sender.**

```
sender's account - who create new account
private key: L5GTSKkRs9NPsXwYgACZdodNUJqCAWjz2BccuR4cAgxJumEZWjokmpr
address: Gu5xHjhos5WkjGo9jKmYMY7dwWWzbEGdQCs11QkyAhh8mca

ac0
public key - weight: cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu - 100
threshold: 100
initial balance: 50 MCC

ac1
public key - weight: sdjgo1jJ2kxAxMyBj6qZDb8okZpwzHYE8ZACgePYW4eTmpu - 100
threshold: 100
initial balance: 50 MCC
```

Then,

```
$ NETWORK_ID=mitum

$ NODE=https://127.0.0.1:54321

$ SENDER_PRV=L5GTSKkRs9NPsXwYgACZdodNUJqCAWjz2BccuR4cAgxJumEZWjokmpr

$ SENDER_ADDR=Gu5xHjhos5WkjGo9jKmYMY7dwWWzbEGdQCs11QkyAhh8mca
```

(continues on next page)

(continued from previous page)

```
$ CURRENCY_ID=MCC

$ AC0_PUB=cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu

$ AC1_PUB=sdjgo1jJ2kxAxMyBj6qZDb8okZpwzHYE8ZACgePYW4eTmpu

$ ./mc seal create-account --network-id=$NETWORK_ID \
    $SENDER_PRV $SENDER_ADDR $CURRENCY_ID,50 \
        --key=$AC0_PUB,100 |
    ./mc seal create-account --network-id=$NETWORK_ID \
        $SENDER_PRV $SENDER_ADDR $CURRENCY_ID,50 \
        --key=$AC1_PUB,100 --seal=- | \
    ./mc seal send --network-id="$NETWORK_ID" \
        $SENDER_PRV --seal=- --node=$NODE --tls-insecure | jq -R '. as $line | try␣
↪fromjson catch $line'
{
    "_hint": "seal-v0.0.1",
    "hash": "HV1tT3D639TiYe6bmamXtesvNjAN8tJ7AmgmeB6STrwz",
    "body_hash": "Gg5KQzzNPAt5PiLrcE5kjMbd4jB7Vk4ooBmN81yWDqYv",
    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
    "signature":
↪"381yXZ1szjaYdxsznCpSvg19yS1tKUw1yPmgXBX6Ehf5ZcKNaMCRkJ8PaNS34rUwLSZ88EPh8vFq1FfRncHiTfo1v9adHCSH
↪",
    "signed_at": "2021-06-10T15:01:13.080144Z",
    "operations": [
        {
            "memo": "",
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "AhqQMGZHDCeJDp74aQJ8rEXMC6GgQtpxP3rXnjjP41ui",
            "fact": {
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "3fDBD1i6V5VpGxB1di6JGgMPhyWZeWRML8FX4LnYXqJE",
                "token": "MjAyMS0wNi0xMFQxNTowMToxMy4wNDA0OTZa",
                "sender": "Gu5xHjhos5WkjGo9jKmYMY7dwWWzbEGdQCs11QkyAhh8mca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLu",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
```

(continues on next page)

(continued from previous page)

```
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "50",
                                "currency": "MCC"
                            }
                        ]
                    },
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "EuCb6BVafkV1tBLsrMqkxojkanJCM4bvmG6JFUZ4s7XL",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"sdjgo1jJ2kxAxMyBj6qZDb8okZpwzHYE8ZACgePYW4eTmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "50",
                                "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
                    "signature":
↪"AN1rKvthtCymTu7gv2fSrMhGwqVuK3o24FrDe6GGLzRU8N5SWF62nPs3iKcEjuzwHya6P9JmrNLRF95ri8QTE4NBc66TxhCHm
↪",
                    "signed_at": "2021-06-10T15:01:13.053303Z"
                }
            ]
        }
    ]
}
"2021-06-10T15:01:13.083634Z INF trying to send seal module=command-send-seal"
"2021-06-10T15:01:13.171266Z INF sent seal module=command-send-seal"
```

Whether the operation block is saved can be checked through the `fact.hash` of operation inquiry in the digest API.

```
$ FACT_HASH=3fDBD1i6V5VpGxB1di6JGgMPhyWZeWRML8FX4LnYXqJE

$ DIGEST_API="https://127.0.0.1:54320"

$ curl --insecure -v $DIGEST_API/block/operation/$FACT_HASH | jq
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-operation-value-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-operation-value-v0.0.1",
        "hash": "3fDBD1i6V5VpGxB1di6JGgMPhyWZeWRML8FX4LnYXqJE",
        "operation": {
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "AhqQMGZHDCeJDp74aQJ8rEXMC6GgQtpxP3rXnjjP41ui",
            "fact": {
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "3fDBD1i6V5VpGxB1di6JGgMPhyWZeWRML8FX4LnYXqJE",
                "token": "MjAyMS0wNi0xMFQxNTowMToxMy4wNDA0OTZa",
                "sender": "Gu5xHjhos5WkjGo9jKmYMY7dwWWzbEGdQCs11QkyAhh8mca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLu",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "50",
                                "currency": "MCC"
                            }
                        ]
                    },
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "EuCb6BVafkV1tBLsrMqkxojkanJCM4bvmG6JFUZ4s7XL",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
```

(continues on next page)

(continued from previous page)

```
↪"sdjgo1jJ2kxAxMyBj6qZDb8okZpwzHYE8ZACgePYW4eTmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "50",
                                "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
                    "signature":
↪"AN1rKvthtCymTu7gv2fSrMhGwqVuK3o24FrDe6GGLzRU8N5SWF62nPs3iKcEjuzwHya6P9JmrNLRF95ri8QTE4NBc66TxhCHm
↪",
                    "signed_at": "2021-06-10T15:01:13.053Z"
                }
            ],
            "memo": ""
        },
        "height": 13,
        "confirmed_at": "2021-06-10T15:01:13.354Z",
        "reason": null,
        "in_state": true,
        "index": 0
    },
    "_links": {
        "block": {
            "href": "/block/13"
        },
        "manifest": {
            "href": "/block/13/manifest"
        },
        "new_account:8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLu": {
            "href": "/account/8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
            "address": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
            "key": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLu"
        },
        "new_account:EuCb6BVafkV1tBLsrMqkxojkanJCM4bvmG6JFUZ4s7XL": {
            "href": "/account/2S252hnemi1oA3UZqEA7dvMSvbd3RA9ut1mgJNxoGW1Pmca",
            "key": "EuCb6BVafkV1tBLsrMqkxojkanJCM4bvmG6JFUZ4s7XL",
            "address": "2S252hnemi1oA3UZqEA7dvMSvbd3RA9ut1mgJNxoGW1Pmca"
        },
        "operation:{hash}": {
            "templated": true,
```

(continues on next page)

(continued from previous page)

```
            "href": "/block/operation/{hash:(?i)[0-9a-z][0-9a-z]+}"
        },
        "block:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "self": {
            "href": "/block/operation/3fDBD1i6V5VpGxB1di6JGgMPhyWZeWRML8FX4LnYXqJE"
        }
    }
}
```

2.11.2 transfer

The `transfer` command is used for transferring tokens between accounts.

```
$ ./mc seal transfer --network-id=NETWORK-ID-FLAG <privatekey> <sender> <receiver>
↪<currency,amount> ...
```

EXAMPLE

This is an example of transferring the currency 10 *MCC* tokens from `ac0` to `ac1`.

```
$ AC0_PRV=KzUYFHNzxvUnZfm1ePJJ4gnLcLtMv1Tvod7Fib2sRuFmGwzm1GVbmpr

$ AC0_ADDR=FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca

$ AC1_ADDR=HjyXhhuHAZBGaEw2S5cKZhDwqVc1StbkJMtdgGm3F1dnmca

$ CURRENCY_ID=MCC

$ NETWORK_ID="mitum"

$ ./mc seal transfer --network-id=$NETWORK_ID $AC0_PRV $AC0_ADDR $AC1_ADDR $CURRENCY_ID,
↪10 | jq
{
    "_hint": "seal-v0.0.1",
    "hash": "EJDzHbusvvcknN9NWaK1wjuvSTav2TVfnDmtRnqVjEVn",
    "body_hash": "FWLTyQePguo6CFxH8SgEHesoLL8ab3FofEw9nXHDDLMp",
    "signer": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu",
    "signature":
↪"381yXZMbRqwMgfWwJNk4rWNuaJenJMHZU3HBufz7Uo4Yj3zo944oeJeGoKjUDyCJXuL4pZLt49gqW2FHV3YuB5zBR24h96ZH
↪",
    "signed_at": "2021-06-14T03:42:11.969679Z",
    "operations": [
```

(continues on next page)

(continued from previous page)

```
        {
            "_hint": "mitum-currency-transfers-operation-v0.0.1",
            "hash": "F3WZYRgcwwYENiVXx6J6zKPqkiDjSZcuF2vUUPiyR3n9",
            "fact": {
                "_hint": "mitum-currency-transfers-operation-fact-v0.0.1",
                "hash": "7xzioXfnkKU1qrFvgeWK1KrhR71RMHMSBZdpWRVK3MUD",
                "token": "MjAyMS0wNi0xNFQwMzo0MjoxMS45NjUyNjJa",
                "sender": "FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
                "items": [
                    {
                        "_hint": "mitum-currency-transfers-item-single-amount-v0.0.1",
                        "receiver": "HjyXhhuHAZBGaEw2S5cKZhDwqVc1StbkJMtdgGm3F1dnmca",
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "10",
                                "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu",
                    "signature":
↪"AN1rKvtRQeMWcFQ9oPLqgakgW33fed4mCcxxfQwi3icWLyn19AKJ3XpYehA8njvAi7qzgGSVpv23JXBDcXbwiZvQkHBj6T8jw
↪",
                    "signed_at": "2021-06-14T03:42:11.96891Z"
                }
            ],
            "memo": ""
        }
    ]
}
```

If you want to send the operation to the network right away,

```
$ ./mc seal transfer --network-id=$NETWORK_ID $AC0_PRV $AC0_ADDR $AC1_ADDR $CURRENCY_ID,
↪3 | jq \
    ./mc seal send --network-id=$NETWORK_ID $AC0_PRV --seal=-
```

2.11.3 key-updater

The `key-updater` command is used for updating the account keys.

Updating account keys to new public keys does not change the address.

```
$ ./mc seal key-updater --network-id=NETWORK-ID-FLAG <privatekey> <target> <currency> --
↪key=KEY@... --threshold=THRESHOLD
```

- KEY: <pub key, weight>

For more information about account keys, refer to *Multi Sig Account*.

EXAMPLE

This is an example of `key-updater`. The example shows updating keys of `ac0` to another one.

```
ac0 - target account
private key: KzUYFHNzxvUnZfm1ePJJ4gnLcLtMv1Tvod7Fib2sRuFmGwzm1GVbmpr
public key: 2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu
address: FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca

ac1 - new key
public key: 247KCJyus9NYJii9rkT4R3z6GxengcwYQHwRKA6DySbiUmpu
```

```
$ NETWORK_ID="mitum"

$ NODE=https://127.0.0.1:54321

$ AC0_PRV=KzUYFHNzxvUnZfm1ePJJ4gnLcLtMv1Tvod7Fib2sRuFmGwzm1GVbmpr

$ AC0_PUB=2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu

$ AC0_ADDR=FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca

$ AC1_PUB=247KCJyus9NYJii9rkT4R3z6GxengcwYQHwRKA6DySbiUmpu

$ CURRENCY_ID=MCC

$ ./mc seal key-updater --network-id=$NETWORK_ID $AC0_PRV $AC0_ADDR --key $AC1_PUB,100
↪$CURRENCY_ID | jq
{
    "_hint": "seal-v0.0.1",
    "hash": "GvuGxKCTKWqXzgzxk3iWVGkSPAMn1nBNbAu7qgzHB8y6",
    "body_hash": "8gyB4eE7yQvneA463ZnM8LEWKDCthm8mKEFcfvAmk2pg",
    "signer": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu",
    "signature":
↪"381yXZWCaZy3G5VLse9NCBMmJg8bPWoY4rmyAWMTRVjLKZP9WkexgJfN8EP4G2P64MPchFKtsYZ2QsNyu31rrjKQN4THtEtz
```

(continues on next page)

(continued from previous page)

```
↪",
    "signed_at": "2021-06-14T03:45:21.821896Z",
    "operations": [
        {
            "_hint": "mitum-currency-keyupdater-operation-v0.0.1",
            "hash": "4fFKpjDBmSrka3C3Q62fz5JYGZstZmkQTe27vgyNj4A9",
            "fact": {
                "_hint": "mitum-currency-keyupdater-operation-fact-v0.0.1",
                "hash": "5yaMz2aSKS5H1wtd4YVcU4q5awbaxu7bhhswX3ss8XCb",
                "token": "MjAyMS0wNi0xNFQwMzo0NToyMS44MTczNjNa",
                "target": "FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
                "keys": {
                    "_hint": "mitum-currency-keys-v0.0.1",
                    "hash": "GmUiuEbsoTVLSirRWMZ2WcxT69enhEXNfskAnRJby8he",
                    "keys": [
                        {
                            "_hint": "mitum-currency-key-v0.0.1",
                            "weight": 100,
                            "key": "247KCJyus9NYJii9rkT4R3z6GxengcwYQHwRKA6DySbiUmpu"
                        }
                    ],
                    "threshold": 100
                },
                "currency": "MCC"
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu",
                    "signature":
↪"AN1rKvtPv6CuiW36Q4g1wtmsGNy2Fc3ierpHgfnjXjdqjDE3wvSH293FVDYy9Yf9VTNadfMGJ38WC39hthZuGkau3vBGq7ijP
↪",
                    "signed_at": "2021-06-14T03:45:21.821399Z"
                }
            ],
            "memo": ""
        }
    ]
}
```

If you want to send the operation right away,

```
$ ./mc seal key-updater --network-id=$NETWORK_ID $AC0_PRV $AC0_ADDR \
    --key $AC1_PUB,100" $CURRENCY_ID \
    | ./mc seal send --network-id=$NETWORK_ID \
    $AC0_PRV --seal=- --node=$NODE --tls-insecure
```

Also, you can check whether the account keys have really changed.

```
$ find blockfs -name "*-states-*" -print | sort -g | xargs -n 1 gzcat |  grep '^{' | jq
↪'. | select(.key == "'$ACO_ACC_KEY'") | [ "height: "+(.height|tostring),   "state_key:
↪" + .key, "key.publickey: " + .value.value.keys.keys[0].key, "key.weight: " + (.value.
↪value.keys.keys[0].weight|tostring), "threshold: " + (.value.value.keys.
↪threshold|tostring)]'
[
    "height: 3",
    "state_key: GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623-mca:account",
    "key.publickey: 2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu",
    "key.weight: 100",
    "threshold: 100"
]
[
    "height: 104",
    "state_key: GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623-mca:account",
    "key.publickey: 247KCJyus9NYJii9rkT4R3z6GxengcwYQHwRKA6DySbiUmpu",
    "key.weight: 100",
    "threshold: 100"
]
```

2.11.4 currency-register

The `currency-register` command is used for registering a new currency token.

```
$ ./mc seal currency-register --network-id=NETWORK-ID-FLAG --feeer=STRING <privatekey>
↪<currency-id> <genesis-amount> <genesis-account>
```

When registering a new currency, the items that need to be set are as follows.

- `genesis account`: account where the issued token will be registered with new currency registration
- `genesis amount`: amount of newly issued tokens
- `-policy-new-account-min-balance=<amount>` must be set.
- `feeer`: The feeer can be selected from three policies; {nil, fixed, ratio}.
  - `nil` is a case where there is no fee payment.
  - `fixed` is a case where a fixed amount is paid.
  - `ratio` is a case where a payment is made in proportion to the operation amount.
  - If the fee policy is fixed, you must set `-feeer-fixed-receiver=<fee receiver account address>` and `-feeer-fixed-amount=<fee amount>` accordingly.
  - If the fee policy is ratio, then `-feeer-ratio-receiver=<fee receiver account address>` and `-feeer-ratio-ratio=<fee ratio, multifly by operation amount>,`` –feeer-ratio-min=<minimum fee>``,`` –feeer-ratio-max=<maximum fee>`` must be set.

When registering a new currency, **the signature of the suffrage nodes participating in consensus must exceed the consensus threshold (67%) to be executed**.

EXAMPLE

Suppose that we are going to register a new currency **MCC2** with the following conditions.

```
genesis-account : ac1
genesis-amount : 9999999999999
currency-id : MCC2
feeer : fixed
feeer-fixed-receiver : ac1
feeer-fixed-amount : 3
seal sender : ac1
suffrage node : n0, n1, n2, n3
```

Then,

```
$ NETWORK_ID="mitum"

$ AC1_ADDR="HWXPq5mBSneSsQis6BbrNT6nvpkafuBqE6F2vgaTYfAC-a000:0.0.1"

$ AC1_PRV="792c971c801a8e45745938946a85b1089e61c1cdc310cf61370568bf260a29be-0114:0.0.1"

$ N0_PRV=<n0 private key>

$ N1_PRV=<n1 private key>

$ N2_PRV=<n2 private key>

$ N3_PRV=<n3 private key>

$ ./mc seal currency-register --network-id=$NETWORK_ID --feeer=fixed --feeer-fixed-
↪receiver=$AC1_ADDR \
    --feeer-fixed-amount=3 --policy-new-account-min-balance=10 $N0_PRV MCC2␣
↪9999999999999 $AC1_ADDR \
    | ./mc seal sign-fact $N1_PRV --network-id="$NETWORK_ID" --seal=- \
    | ./mc seal sign-fact $N2_PRV --network-id="$NETWORK_ID" --seal=- \
    | ./mc seal sign-fact $N3_PRV --network-id="$NETWORK_ID" --seal=- \
    | ./mc seal send --network-id="$NETWORK_ID" $AC1_PRV --seal=-
```

Each currency has a *zero account* for deposit only that is used to **burn tokens**. The *zero account* is deposit only because the public key is not registered.

The address of the *zero account* has the same format as `<currency id>-Xmca`. For example, the *zero account* address of PEN currency is `PEN-Xmca`.

```
$ curl --insecure http://localhost:54320/account/PEN-Xmca | jq
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-account-value-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-account-value-v0.0.1",
        "hash": "EJvkxncxfVQNncdKZtjQTH2XuT5ECRiqSZA7LLE14zqi",
        "address": "PEN-Xmca",
        "keys": {
            "_hint": "mitum-currency-keys-v0.0.1",
            "hash": "",
            "keys": [],
            "threshold": 0
        },
        "balance": [
            {
                "_hint": "mitum-currency-amount-v0.0.1",
                "amount": "100000000000000000000000000",
                "currency": "PEN"
            }
        ],
        "height": 41,
        "previous_height": 0
    },
    "_links": {
        "block": {
            "href": "/block/41"
        },
        "previous_block": {
            "href": "/block/0"
        },
        "self": {
            "href": "/account/PEN-Xmca"
        },
        "operations": {
            "href": "/account/PEN-Xmca/operations"
        },
        "operations:{offset}": {
            "href": "/account/PEN-Xmca/operations?offset={offset}",
            "templated": true
        },
        "operations:{offset,reverse}": {
            "templated": true,
            "href": "/account/PEN-Xmca/operations?offset={offset}&reverse=1"
        }
    }
}
```

2.11.5 currency-policy-updater

The `currency-policy-updater` command is used for updating the currency-related policy.

```
$ ./mc seal currency-policy-updater --network-id=NETWORK-ID-FLAG --feeer=STRING
↪<privatekey> <currency-id>
```

First, get the info of the registered currency through API.

When updating a currency policy, **the signature of the suffrage nodes participating in consensus must exceed the consensus threshold (67%) to be executed**.

```
$ curl --insecure -v https://localhost:54320/currency/MCC2 | jq
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-currency-design-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-currency-design-v0.0.1",
        "amount": {
            "_hint": "mitum-currency-amount-v0.0.1",
            "amount": "9999999999999",
            "currency": "MCC2"
        },
        "genesis_account": "FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
        "policy": {
            "_hint": "mitum-currency-currency-policy-v0.0.1",
            "new_account_min_balance": "10",
            "feeer": {
                "_hint": "mitum-currency-fixed-feeer-v0.0.1",
                "type": "fixed",
                "receiver": "FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
                "amount": "3"
            }
        }
    },
    "_links": {
        "self": {
            "href": "/currency/MCC2"
        },
        "currency:{currencyid}": {
            "templated": true,
            "href": "/currency/{currencyid:.*}"
        },
        "block": {
            "href": "/block/10"
        },
        "operations": {
            "href": "/block/operation/goNANpmA1BcnXA6TVL6AKkoxsmiaT2F5ss5zoSh7Wdt"
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

The policy that can be changed through `currency-policy-updater` is the **fee-related policy** and the **minimum balance value** when creating a new account.

EXAMPLE

Suppose that we are going to update policy for *MCC2* according to the following conditions.

```
currency-id : MCC2

Policy to be updated
- feeer : ratio
- feeer-ratio-receiver : ac1
- feeer-ratio-ratio : 0.5
- feeer-ratio-min : 3
- feeer-ratio-max : 1000
- policy-new-account-min-balance : 100

suffrage node : n0, n1, n2, n3
```

Then,

```
$ NETWORK_ID="mitum"

$ AC1_ADDR="HjyXhhuHAZBGaEw2S5cKZhDwqVc1StbkJMtdgGm3F1dnmca"

$ AC0_PRV="KzUYFHNzxvUnZfm1ePJJ4gnLcLtMv1Tvod7Fib2sRuFmGwzm1GVbmpr"

$ N0_PRV=<n0 private key>

$ N1_PRV=<n1 private key>

$ N2_PRV=<n2 private key>

$ N3_PRV=<n3 private key>

$ ./mc seal currency-policy-updater --network-id=$NETWORK_ID --feeer="ratio" --feeer-
↪ratio-receiver=$AC1_ADDR \
    --feeer-ratio-ratio=0.5 --feeer-ratio-min=3 --feeer-ratio-max=1000 --policy-new-
↪account-min-balance=100 $N0_PRV MCC2 \
    | ./mc seal sign-fact $N1_PRV --network-id=$NETWORK_ID --seal=- \
```

(continues on next page)

(continued from previous page)

```
    | ./mc seal sign-fact $N2_PRV --network-id=$NETWORK_ID --seal=- \
    | ./mc seal sign-fact $N3_PRV --network-id=$NETWORK_ID --seal=- \
    | ./mc seal send --network-id=$NETWORK_ID $AC0_PRV --seal=-
```

Check,

```
$ curl --insecure https://localhost:54320/currency/MCC2 | jq
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-currency-design-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-currency-design-v0.0.1",
        "amount": {
            "_hint": "mitum-currency-amount-v0.0.1",
            "amount": "9999999999999",
            "currency": "MCC2"
        },
        "genesis_account": "FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
        "policy": {
            "_hint": "mitum-currency-currency-policy-v0.0.1",
            "new_account_min_balance": "100",
            "feeer": {
                "_hint": "mitum-currency-ratio-feeer-v0.0.1",
                "type": "ratio",
                "receiver": "HjyXhhuHAZBGaEw2S5cKZhDwqVc1StbkJMtdgGm3F1dnmca",
                "ratio": 0.5,
                "min": "3",
                "max": "1000"
            }
        }
    },
    "_links": {
        "currency:{currencyid}": {
            "href": "/currency/{currencyid:.*}",
            "templated": true
        },
        "block": {
            "href": "/block/13"
        },
        "operations": {
            "href": "/block/operation/3HxC5VP5Fjzent7uVVLsK44i1tp8ooH4f2Vh4c4uWM4e"
        },
        "self": {
            "href": "/currency/MCC2"
        }
    }
}
```

2.11.6 suffrage-inflation

The `suffrage-inflation` command is used for inflating the supply of an existing currency token.

```
$ ./mc seal suffrage-inflation --network-id=NETWORK-ID-FLAG <privatekey> <inflation item>
↪ ...
```

- `inflation item`: <receiver-account>,<currency-id>,<inflation-amount>

There are two processes of registering a currency in Mitum Currency.

- Through initial genesis currency creation
- By registering a new currency while the network is alive

The registered currency has a total supply amount. The Mitum Currency may increase the amount of tokens in addition to the total supply amount.

When generating new amounts, the items that need to be set are as follows.

- `receiver-account` which receives account of additionally generated tokens.

When inflating a currency, **the signature of the suffrage nodes participating in consensus must exceed the consensus threshold (67%) to be executed**.

EXAMPLE

We are going to inflate the supply of `MCC` according to the following conditions.

```
operation-sender-account : ac1
receiver-account : ac2
inflation-amount : 9999999999999
currency-id : MCC
seal sender : ac1
suffrage node : n0, n1, n2, n3
```

Then,

```
$ NETWORK_ID="mitum"

$ AC1_PRV="L2Q4PqxrhgS39jgGoXsV92LaCHRF2SqTLRwMhCC6Q6in9Vb19aDLmpr"
```

(continues on next page)

(continued from previous page)

```
$ AC2_ADDR="HjyXhhuHAZBGaEw2S5cKZhDwqVc1StbkJMtdgGm3F1dnmca"

$ N0_PRV=<n0 private key>

$ N1_PRV=<n1 private key>

$ N2_PRV=<n2 private key>

$ N3_PRV=<n3 private key>

$ ./mc seal suffrage-inflation --network-id=$NETWORK_ID $N0_PRV MCC 9999999999999 $AC2_
↪ADDR \
    | ./mc seal sign-fact $N1_PRV --network-id=$NETWORK_ID --seal=- \
    | ./mc seal sign-fact $N2_PRV --network-id=$NETWORK_ID --seal=- \
    | ./mc seal sign-fact $N3_PRV --network-id=$NETWORK_ID --seal=- \
    | ./mc seal send --network-id=$NETWORK_ID $AC1_PRV --seal=-
```

2.11.7 send

The send command is used for sending a seal.

```
$ ./mc seal send  <sender privatekey> --network-id=<network id> --seal=<data file path> -
↪-node=<node https url>
```

Operations created in Mitum Currency are **transmitted in units of seals**.

Signature is required to transmit the seal. Refer to *Seal* for the part related to the keypair used for signature creation.

EXAMPLE

data.json is a seal file written in json.

```
$ NETWORK_ID="mitum"

$ NODE="https://127.0.0.1:54321"

$ AC0_PRV=L1jPsE8Sjo5QerUHJUZNRqdH1ctxTWzc1ue8Zp2mtpieNwtCKsNZmpr

$ ./mc seal send --network-id=$NETWORK_ID $AC0_PRV --seal=data.json --node=$NODE jq -R '.
↪ as $line | try fromjson catch $line'
{
    "_hint": "seal-v0.0.1",
```

(continues on next page)

(continued from previous page)

```
    "hash": "6nLRWj5hGQ7va9gxpAJCBxNDKvgFnms9jaa913uWgsx1",
    "body_hash": "32ZEf8V9fV41JHVWbbqQdYWtrw5T255XN8fSXhBAhGFD",
    "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
    "signature":
↪"381yXZ4LFY5HnK211gpG3W22V52vMLqix4SysXEeMnqcXUk5eEYGM1JfFaX5UE86EF6qog5jUScPqZo6UkiaAFocUhwtSsjx
↪",
    "signed_at": "2021-06-10T09:17:51.236729Z",
    "operations": [
        {
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "7YvcA6WAcKEag1Z4Jv1bQ2wYxAZix5sNB6u8MUXDM44D",
            "fact": {
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "3equMRJAVHk8WdVanffzEWkHfwnBDqF2cFwmmcv8MzDW",
                "token": "MjAyMS0wNi0xMFQwOToxNzo1MS4yMDgwOTVa",
                "sender": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "100000",
                                "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
                    "signature":
↪"AN1rKvtPEX4MRu6kWRYDJ6WtsSnwxwJsYXiVi2Qujx8sF6SJzsZZKj7anCd9cmUZ175FSYLkkWkpDRj3fVgZFDxLFSnos3szz
↪",
                    "signed_at": "2021-06-10T09:17:51.211816Z"
                }
            ],
```

(continues on next page)

(continued from previous page)

```
            "memo": ""
        }
    ]
}
2021-06-10T09:17:51.240066Z INF trying to send seal module=command-send-seal
2021-06-10T09:17:51.345243Z INF sent seal module=command-send-seal
```

When sending to a local node for testing, an error may occur related to tls authentication.

In this case, give the option `-tls-insecure=true` when sending seals.

```
$ ./mc seal send --network-id=$NETWORK_ID $AC0_PRV --tls-insecure=true --seal=data.json -
↪-node=$NODE
```

2.11.8 sign

The `sign` command is used for creating a signature for a seal.

```
$ ./mc seal sign --network-id=NETWORK-ID-FLAG <privatekey>
```

EXAMPLE

Before using the `sign` command, prepare a file containing a seal with operations. The file should be saved in json format for signature generation.

For example,

```
{
    "_hint": "seal-v0.0.1",
    "hash": "5W39B2mmtc4KK9THiRdoF6F5UMZPSxjzedPePojVhqyV",
    "body_hash": "5yGtCzJiPRRbZkeLawQev4dvdYgYuKHXe6TP6x2VLSt4",
    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
    "signature":
↪"381yXZHsyzbc8qTD7BJgmGoM8ncSrUcyDZiSNanARp9h84tvcj6HkGXzpFyck9arJTCQDmPGzT5UFq1coHv7wijusgynSfgr
↪",
    "signed_at": "2021-06-10T06:50:26.903245Z",
    "operations": [
        {
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "9mFHaqd66pv7RjoAbKScUucJLKW7KVSkWqN1WXnzMrxQ",
            "fact": {
```

(continues on next page)

(continued from previous page)

```
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "3CpL1MgD1TPejUmVxPKSgiUu6LCR7FhFrDehSjSogavZ",
                "token": "MjAyMS0wNi0xMFQwNjo1MDoyNi44NzQyNzVa",
                "sender": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "Dut3WiprEo1BRcx2xRvh6qbBgxaTLXQDris7SihDTET8",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"27tMvbSpajF1VSnrn3xRQESpPAsmA7KZEfUz9ZuTZEemumpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "100000",
                                "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
                    "signature":
↪"AN1rKvtfRrgY15owfURsNyfWnYtZ7syuRafWa637tkWB1HyxDCD2tWZUhySTg6mnZWQKpP3i6Dmf96fw9TUWb8rrbsetHJciH
↪",
                    "signed_at": "2021-06-10T06:50:26.877954Z"
                }
            ],
            "memo": ""
        }
    ]
}
```

Run `seal sign` with this json file.

Then, you can get a seal with a new seal signature, as shown in the following.

```
$ SIGNER_PRV=KxmWM4Zj5Ln8bbDwVZEKrYQY8N51Uk3UVq5GNQAeb2KW8JqHmsgmmpr
$ ./mc seal sign --seal=data.json  --network-id=mitum $SIGNER_PRV | jq
{
    "_hint": "seal-v0.0.1",
    "hash": "5dLCySkPrFtc8SnbjzELBK5GR7VQocrK7cXswEnhEa1S",
    "body_hash": "3Ah7J2q4HhFXSgV3c4EQWeZtpi1nFY7be2nmL4X6qDxa",
    "signer": "224ekkhrax6EpekzfLTv9See1hNDZW3LAjWBRuzTMpgnrmpu",
    "signature":
↪"AN1rKvtFhZfDzyLLXtK3PtZ8P1jSTqZy6gC8WooBjWRhzwLrXjCcVTeo4juzdMg83he2emJ3SVkCNZssiB1pTtAPtx753P5CT
↪",
    "signed_at": "2021-06-10T07:12:41.992205Z",
    "operations": [
        {
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "9mFHaqd66pv7RjoAbKScUucJLKW7KVSkWqN1WXnzMrxQ",
            "fact": {
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "3CpL1MgD1TPejUmVxPKSgiUu6LCR7FhFrDehSjSogavZ",
                "token": "MjAyMS0wNi0xMFQwNjo1MDoyNi44NzQyNzVa",
                "sender": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "Dut3WiprEo1BRcx2xRvh6qbBgxaTLXQDris7SihDTET8",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"27tMvbSpajF1VSnrn3xRQESpPAsmA7KZEfUz9ZuTZEemumpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                            "_hint": "mitum-currency-amount-v0.0.1",
                            "amount": "100000",
                            "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
                    "signature":
↪"AN1rKvtfRrgY15owfURsNyfWnYtZ7syuRafWa637tkWB1HyxDCD2tWZUhySTg6mnZWQKpP3i6Dmf96fw9TUWb8rrbsetHJciH
↪",
```

(continues on next page)

(continued from previous page)

```
                    "signed_at": "2021-06-10T06:50:26.877954Z"
                }
            ],
            "memo": ""
        }
    ]
}
```

2.11.9 sign-fact

The `sign-fact` command is used for creating signatures for operation facts.

This command is used to add a fact signature to the operation contained in the seal. You must pass the seal data containing the operation to this command.

This command is mainly used when an operation is created by an account with multi sig or when signing multiple nodes is required, such as in currency registration.

```
$ ./mc seal sign-fact --network-id=NETWORK-ID-FLAG <privatekey>
```

EXAMPLE

Here is an example where a seal contains a transfer operation for transferring tokens from the multi sig account. It requires two fact signatures, but only has one.

```
{
    "_hint": "seal-v0.0.1",
    "hash": "CgFaHkJEP966xRQjzPtXBUwzqgQYWB53RHwjBqyvmKHs",
    "body_hash": "Akjx1kJZKzyYMo2eVbqcUvtEfivDEGsK4yeUUuNwbGmu",
    "signer": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu",
    "signature":
↪"381yXZ8qZBYQXDBaGr1KyAcsMJyB9HZLo1aQQRsxhx854aMYm5n7nh3NXzsJHpEhiYHgWUYnCtbAZaVsQ8pe6nEnLaHCXizY
↪",
    "signed_at": "2021-06-10T09:54:35.868873Z",
    "operations": [
        {
            "hash": "Eep8SJH7Vkqft3BcvKYd9NY14Zgzmhyp7Uts2GmpaS5N",
            "fact": {
                "_hint": "mitum-currency-transfers-operation-fact-v0.0.1",
                "hash": "Eu1b4gr528Xy4u2sg97DsEo5uj9BuQEMjHzJxdsLgH48",
                "token": "MjAyMS0wNi0xMFQwOTo1NDozNS44NjQwOTha",
                "sender": "FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
                "items": [
```

(continues on next page)

(continued from previous page)

```
                    {
                        "_hint": "mitum-currency-transfers-item-single-amount-v0.0.1",
                        "receiver": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
                        "amounts": [
                            {
                            "_hint": "mitum-currency-amount-v0.0.1",
                            "amount": "100",
                            "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu",
                    "signature":
↪"AN1rKvtZFkx5e4NexvBSjjJkuzUj45UKau8DL2JZx5d1htnbnkmPmHnNbgwqfvUnz8KHpUR72Z9YxD4JVQhdh4JCzGv9zMDDG
↪",
                    "signed_at": "2021-06-10T09:54:35.868223Z"
                }
            ],
            "memo": "",
            "_hint": "mitum-currency-transfers-operation-v0.0.1"
        }
    ]
}
```

After using `sign-fact` to add a fact signature, the above json becomes,

```
$ SIGNER1_PUB_KEY=2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu
$ SIGNER2_PUB_KEY=sdjgo1jJ2kxAxMyBj6qZDb8okZpwzHYE8ZACgePYW4eTmpu
$ SIGNER2_PRV_KEY=L5AAoEqwnHCp7WfkPcUmtUX61ppZQww345rEDCwB33jVPud4hzKJmpr
$ NETWORK_ID=mitum
$ ./mc seal sign-fact $SIGNER2_PRV_KEY --seal data.json --network-id=$NETWORK_ID | jq

{
    "_hint": "seal-v0.0.1",
    "hash": "GiADUurx7qVwyeu8XUNQgmNpqmtN9UDzockhLNKXzYN6",
    "body_hash": "Ci7yzpahGtXqpWs3EGfoqnmUhTgbRhdkgb2GupsJRvgB",
    "signer": "sdjgo1jJ2kxAxMyBj6qZDb8okZpwzHYE8ZACgePYW4eTmpu",
    "signature":
↪"381yXYnDDMYrZ4asLpAYgD7AHDAGMsVih11S3V2jCwNdvJJxeA96whPnth4DxXoJ3RiK8vBpvVKRvXJsPpDpZZ2GMagAmaBi
↪",
    "signed_at": "2021-06-10T10:01:27.690429Z",
    "operations": [
        {
            "_hint": "mitum-currency-transfers-operation-v0.0.1",
```

(continues on next page)

(continued from previous page)

```
            "hash": "AduowWC9mHTCeRp8aqN4dQxHjKGH8xdm8vqxcMj7SfUZ",
            "fact": {
                "_hint": "mitum-currency-transfers-operation-fact-v0.0.1",
                "hash": "Eu1b4gr528Xy4u2sg97DsEo5uj9BuQEMjHzJxdsLgH48",
                "token": "MjAyMS0wNi0xMFQwOTo1NDozNS44NjQwOTha",
                "sender": "FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
                "items": [
                    {
                        "_hint": "mitum-currency-transfers-item-single-amount-v0.0.1",
                        "receiver": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "100",
                                "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu",
                    "signature":
↪"AN1rKvtZFkx5e4NexvBSjjJkuzUj45UKau8DL2JZx5d1htnbnkmPmHnNbgwqfvUnz8KHpUR72Z9YxD4JVQhdh4JCzGv9zMDDG
↪",
                    "signed_at": "2021-06-10T09:54:35.868223Z"
                },
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "sdjgo1jJ2kxAxMyBj6qZDb8okZpwzHYE8ZACgePYW4eTmpu",
                    "signature":
↪"381yXZ9yqzCSzUZZUuQvU3ZMHgM9Pa5MQUo2hKGhPFW4ZuMCC3eK2iGYvx3gwQD3LCfELuUXejAQiMmeKaNAEoZVPDf1gpkE
↪",
                    "signed_at": "2021-06-10T10:01:27.690034Z"
                }
            ],
            "memo": ""
        }
    ]
}
```

2.12 Storage Command

The `storage` command helps **download**, **verify**, and **restore** block data.

The subcommands related to `storage` command are as follows.

- `download`
- `verify-blockdata`
- `verify-database`
- `clean`
- `clean-by-height`
- `restore`
- `set-blockdatamaps`

2.12.1 download

The `download` command is used for downloading block data of specific blockheight.

```
$ ./mc storage download --node=quic://localhost:54321 <data type> <height> ...
```

EXAMPLE

```
$ ./mc storage download --tls-insecure --node=https://127.0.0.1:54321  --save=data all --
↪ -1 0 1 2 3 4 5
2021-06-08T10:50:08.018561Z INF saved file=data/000/000/000/000/000/000/0_1/-1-manifest-
↪48cfbadd18b892bfd0a6fa230ff0c5f719bd517d37f594012aeca7244ef12599.jsonld.gz height=-1␣
↪module=command-block-download
2021-06-08T10:50:08.018531Z INF saved file=data/000/000/000/000/000/000/000/0-manifest-
↪307ffa78d4ce5e32e25347f5ec8ee626e44d41e55f565c2082ac00f8f128dbd9.jsonld.gz height=0␣
↪module=command-block-download
2021-06-08T10:50:08.058628Z INF saved file=data/000/000/000/000/000/000/0_1/-1-
↪operations-0fedf0c3ccb08aea5694e04a382ca04fb1338dfc9c2c408fe6296c93c0931124.jsonld.gz␣
↪height=-1 module=command-block-download
2021-06-08T10:50:08.068871Z INF saved file=data/000/000/000/000/000/000/000/0-operations-
↪d17d5b941aec3c100a43e2c228bca4134473bb9c78dcf567bdd8b9e12e5cc928.jsonld.gz height=0␣
↪module=command-block-download
2021-06-08T10:50:08.12423Z INF saved file=data/000/000/000/000/000/000/000/0-operations_
↪tree-45aff89f7084384fdecfac9689b75168a33f03bf6ba677ad085a6ac8fdf2bd12.jsonld.gz␣
↪height=0 module=command-block-download
2021-06-08T10:50:08.130027Z INF saved file=data/000/000/000/000/000/000/0_1/-1-
↪operations_tree-d0c45c5292593853052aba6d3f410c93f6cc4473e7873ded2d623069adfc0025.
↪jsonld.gz height=-1 module=command-block-download
2021-06-08T10:50:08.162735Z INF saved file=data/000/000/000/000/000/000/000/0-states-
```

(continues on next page)

(continued from previous page)

```
↪73ac164e67fb49877b132aaaae2f7adf92cc237ef0e63db30f3013c283fb7100.jsonld.gz height=0␣
↪module=command-block-download
2021-06-08T10:50:08.172536Z INF saved file=data/000/000/000/000/000/000/0_1/-1-states-
↪0fedf0c3ccb08aea5694e04a382ca04fb1338dfc9c2c408fe6296c93c0931124.jsonld.gz height=-1␣
↪module=command-block-download
2021-06-08T10:50:08.215233Z INF saved file=data/000/000/000/000/000/000/000/0-states_
↪tree-7155e9c9f393943429f9341f22cba749203eaa2effd51bbbdb9b97c899cac62e.jsonld.gz␣
↪height=0 module=command-block-download
2021-06-08T10:50:08.217385Z INF saved file=data/000/000/000/000/000/000/0_1/-1-states_
↪tree-d0c45c5292593853052aba6d3f410c93f6cc4473e7873ded2d623069adfc0025.jsonld.gz␣
↪height=-1 module=command-block-download
2021-06-08T10:50:08.278019Z INF saved file=data/000/000/000/000/000/000/000/0-init_
↪voteproof-dab53369d715fc74ad750d95f1ceb859d62009165a76ea3368399da2b16bf4d7.jsonld.gz␣
↪height=0 module=command-block-download
2021-06-08T10:50:08.287794Z INF saved file=data/000/000/000/000/000/000/0_1/-1-init_
↪voteproof-812c550f7595c4c949d2255217a343864bdd878b09d124235d7db07758620bc7.jsonld.gz␣
↪height=-1 module=command-block-download
2021-06-08T10:50:08.319642Z INF saved file=data/000/000/000/000/000/000/000/0-accept_
↪voteproof-09fd08050476a5d0a343154aaa0325809d721004b49cba303a58300b7415235e.jsonld.gz␣
↪height=0 module=command-block-download
2021-06-08T10:50:08.334284Z INF saved file=data/000/000/000/000/000/000/0_1/-1-accept_
↪voteproof-812c550f7595c4c949d2255217a343864bdd878b09d124235d7db07758620bc7.jsonld.gz␣
↪height=-1 module=command-block-download
2021-06-08T10:50:08.399426Z INF saved file=data/000/000/000/000/000/000/000/0-suffrage_
↪info-038aa59ed7db04c96d11405336c7a2d1cb8ad6df5a18d66f8f3bf2919c6767f8.jsonld.gz␣
↪height=0 module=command-block-download
2021-06-08T10:50:08.591648Z INF saved file=data/000/000/000/000/000/000/0_1/-1-suffrage_
↪info-038aa59ed7db04c96d11405336c7a2d1cb8ad6df5a18d66f8f3bf2919c6767f8.jsonld.gz␣
↪height=-1 module=command-block-download
2021-06-08T10:50:08.613875Z INF saved file=data/000/000/000/000/000/000/000/0-proposal-
↪81c03f9c912591796ae5f3dbaab85bc91d7ca4031413787abb3068c5efa78360.jsonld.gz height=0␣
↪module=command-block-download
2021-06-08T10:50:08.750795Z INF saved file=data/000/000/000/000/000/000/0_1/-1-proposal-
↪812c550f7595c4c949d2255217a343864bdd878b09d124235d7db07758620bc7.jsonld.gz height=-1␣
↪module=command-block-download
```

map

The `download map` command is used for downloading the blockdata map.

See *Block Data* for details.

```
$ ./mc storage download map --node=https://localhost:54321 <height> ...
```

EXAMPLE

```
$ ./mc storage download map --tls-insecure --node=https://127.0.0.1:54321 0 | jq
{
    "_hint": "base-blockdatamap-v0.0.1",
    "hash": "DvYK11jZ8KWafAGPssypdNMRwwXwJJTKeyzTAx4JNnwc",
    "height": 10,
    "block": "AnjD39fpP6cJKVhnSfJxPfQ8sxrVwCrKhm1zWjb38dUS",
    "created_at": "2021-06-10T06:37:42.251Z",
    "items": {
        "accept_voteproof": {
        "type": "accept_voteproof",
        "checksum": "03dd3c2ce852729ff52ec7dcd31a2a1532656fbcea12a28438c3e84c8146c753",
        "url": "file:///000/000/000/000/000/000/010/10-accept_voteproof-
↪03dd3c2ce852729ff52ec7dcd31a2a1532656fbcea12a28438c3e84c8146c753.jsonld.gz"
        },
        "init_voteproof": {
        "type": "init_voteproof",
        "checksum": "70d59dc3e84ddd06d319e9d38d68a976b09a816fbe5a5fdef42f5b80908b0fa0",
        "url": "file:///000/000/000/000/000/000/010/10-init_voteproof-
↪70d59dc3e84ddd06d319e9d38d68a976b09a816fbe5a5fdef42f5b80908b0fa0.jsonld.gz"
        },
        "states": {
        "type": "states",
        "checksum": "d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59",
        "url": "file:///000/000/000/000/000/000/010/10-states-
↪d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59.jsonld.gz"
        },
        "proposal": {
        "type": "proposal",
        "checksum": "ccd31f6627aa3cc6e9768b318f8cfd8e7f371b907f329fb89d692c7aea2ef465",
        "url": "file:///000/000/000/000/000/000/010/10-proposal-
↪ccd31f6627aa3cc6e9768b318f8cfd8e7f371b907f329fb89d692c7aea2ef465.jsonld.gz"
        },
        "suffrage_info": {
        "type": "suffrage_info",
        "checksum": "f8955c57fb4a7dc48e71973af01852008c76ae4bb5487f8d6fccebcc10e5412e",
        "url": "file:///000/000/000/000/000/000/010/10-suffrage_info-
↪f8955c57fb4a7dc48e71973af01852008c76ae4bb5487f8d6fccebcc10e5412e.jsonld.gz"
        },
        "manifest": {
        "type": "manifest",
        "checksum": "1f21552b0d7a11c0397c7429849a0f611d9681f70cecd5165e21fcbd5276a880",
        "url": "file:///000/000/000/000/000/000/010/10-manifest-
↪1f21552b0d7a11c0397c7429849a0f611d9681f70cecd5165e21fcbd5276a880.jsonld.gz"
        },
        "operations": {
        "type": "operations",
        "checksum": "d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59",
        "url": "file:///000/000/000/000/000/000/010/10-operations-
↪d890f3ba40375a6b2d331883907dc0a9ca980ce45f7d5dcaca9087278c0b6d59.jsonld.gz"
        },
        "states_tree": {
        "type": "states_tree",
        "checksum": "1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26",
```

(continues on next page)

(continued from previous page)

```
        "url": "file:///000/000/000/000/000/000/010/10-states_tree-
↪1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz"
        },
        "operations_tree": {
        "type": "operations_tree",
        "checksum": "1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26",
        "url": "file:///000/000/000/000/000/000/010/10-operations_tree-
↪1f9877aebf8854fd42154c6e6479ff6a3e379b2762c65995c80f3dff2a357a26.jsonld.gz"
        }
    },
    "writer": "blockdata-writer-v0.0.1"
}
```

2.12.2 verify-blockdata

The `verify-blockdata` command is used for verifying blockdata in local storage.

```
$ ./mc storage verify-blockdata <blockdata path>
```

EXAMPLE

```
$ ./mc storage verify-blockdata data --network-id=mitum --verbose
2021-06-08T10:52:03.249204Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/cmd.go:86 > maxprocs: Leaving GOMAXPROCS=8:␣
↪CPU quota undefined module=command-blockdata-verify
2021-06-08T10:52:03.250015Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/cmd.go:95 > flags parsed flags={"CPUProf":
↪"mitum-cpu.pprof","EnableProfiling":false,"LogColor":false,"LogFile":null,"LogFormat":
↪"terminal","LogLevel":"info","LogOutput":{},"MemProf":"mitum-mem.pprof","NetworkID":
↪"bWl0dW0=","Path":"data","TraceProf":"mitum-trace.pprof","Verbose":true}␣
↪module=command-blockdata-verify
2021-06-08T10:52:03.250188Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:38 > trying to verify␣
↪blockdata module=command-blockdata-verify path=data
2021-06-08T10:52:03.250315Z INF ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:107 > last height found␣
↪last_height=5 module=command-blockdata-verify
2021-06-08T10:52:03.250607Z INF ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/verify_storage.go:53 > checking manifests␣
↪module=command-blockdata-verify
2021-06-08T10:52:03.255675Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/verify_storage.go:109 > manifests loaded␣
↪heights=[-1,6] module=command-blockdata-verify
2021-06-08T10:52:03.255766Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/verify_storage.go:121 > manifests checked␣
↪heights=[-1,6] module=command-blockdata-verify
```

(continues on next page)

(continued from previous page)

```
2021-06-08T10:52:03.258293Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:257 > block data files␣
↪checked height=0 module=command-blockdata-verify
2021-06-08T10:52:03.257947Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:257 > block data files␣
↪checked height=1 module=command-blockdata-verify
2021-06-08T10:52:03.259131Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:257 > block data files␣
↪checked height=4 module=command-blockdata-verify
2021-06-08T10:52:03.257772Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:257 > block data files␣
↪checked height=5 module=command-blockdata-verify
2021-06-08T10:52:03.260384Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:257 > block data files␣
↪checked height=2 module=command-blockdata-verify
2021-06-08T10:52:03.260419Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:257 > block data files␣
↪checked height=-1 module=command-blockdata-verify
2021-06-08T10:52:03.260606Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:257 > block data files␣
↪checked height=3 module=command-blockdata-verify
2021-06-08T10:52:03.274069Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:187 > block checked␣
↪height=-1 module=command-blockdata-verify
2021-06-08T10:52:03.279165Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:187 > block checked␣
↪height=3 module=command-blockdata-verify
2021-06-08T10:52:03.279179Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:187 > block checked␣
↪height=2 module=command-blockdata-verify
2021-06-08T10:52:03.279223Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:187 > block checked␣
↪height=1 module=command-blockdata-verify
2021-06-08T10:52:03.279267Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:187 > block checked␣
↪height=4 module=command-blockdata-verify
2021-06-08T10:52:03.279344Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:187 > block checked␣
↪height=5 module=command-blockdata-verify
2021-06-08T10:52:03.281481Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:187 > block checked␣
↪height=0 module=command-blockdata-verify
2021-06-08T10:52:03.281569Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/blockdata_verify.go:87 > blockdata verified␣
↪module=command-blockdata-verify
.....
```

2.12.3 verify-database

The `verify-database` command is used for verifying the database by comparing it with the block data.

```
$ ./mc storage verify-database <database uri> <blockdata path>
```

EXAMPLE

```
$ ./mc storage verify-database mongodb://127.0.0.1:27017/n0_mc blockfs --network-
↪id=mitum --verbose
2021-06-08T10:56:20.879671Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/cmd.go:86 > maxprocs: Leaving GOMAXPROCS=8:␣
↪CPU quota undefined module=command-database-verify
2021-06-08T10:56:20.879921Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/cmds/cmd.go:95 > flags parsed flags={"CPUProf":
↪"mitum-cpu.pprof","EnableProfiling":false,"LogColor":false,"LogFile":null,"LogFormat":
↪"terminal","LogLevel":"info","LogOutput":{},"MemProf":"mitum-mem.pprof","NetworkID":
↪"bWl0dW0=","Path":"data","TraceProf":"mitum-trace.pprof","URI":"mongodb://127.0.0.
↪1:27017/mc","Verbose":true} module=command-database-verify
2021-06-08T10:56:20.880018Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/pm/processes.go:310 > processed from_process=␣
↪module=process-manager process=init
2021-06-08T10:56:20.880066Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/launch/pm/processes.go:310 > processed from_process=time-
↪syncer module=process-manager process=config
2021-06-08T10:56:21.038454Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.0-
↪20210605063447-f720096b150d/util/localtime/time_sync.go:67 > started interval=120000␣
↪module=time-syncer server=time.google.com
2021-06-08T10:56:21.042330408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/pm/processes.go:310 > processed from_process=init␣
↪module=process-manager process=time-syncer
2021-06-08T10:56:21.042835408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/pm/processes.go:359 > hook processed␣
↪from=encoders hook=add_hinters module=process-manager
2021-06-08T10:56:21.042884408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/pm/processes.go:310 > processed from_process=init␣
↪module=process-manager process=encoders
2021-06-08T10:56:21.203404408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/pm/processes.go:310 > processed from_process=init␣
↪module=process-manager process=database
2021-06-08T10:56:21.203608408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/pm/processes.go:359 > hook processed␣
↪from=blockdata hook=check_blockdata_path module=process-manager
2021-06-08T10:56:21.203899408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/cmds/database_verify.go:207 > block found block={
↪"hash":"CzF6t6ePyBaz6RnSjw6YRhwKsxA5sRnhHwQJvK8xVgMR","height":0,"round":0}␣
↪module=command-database-verify
2021-06-08T10:56:21.204001408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/pm/processes.go:359 > hook processed␣
```

(continues on next page)

(continued from previous page)

```
↪from=blockdata hook=check_storage module=process-manager
2021-06-08T10:56:21.204054408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/pm/processes.go:310 > processed from_process=init␣
↪module=process-manager process=blockdata
2021-06-08T10:56:21.204357408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/cmds/database_verify.go:74 > trying to verify␣
↪database module=command-database-verify path=data uri=mongodb://127.0.0.1:27017/mc
2021-06-08T10:56:21.204424408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/cmds/database_verify.go:100 > verifying database␣
↪module=command-database-verify
2021-06-08T10:56:21.204941408Z INF ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/cmds/verify_storage.go:53 > checking manifests␣
↪module=command-database-verify
2021-06-08T10:56:21.210215408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/cmds/verify_storage.go:109 > manifests loaded␣
↪heights=[-1,1] module=command-database-verify
2021-06-08T10:56:21.210355408Z DBG ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/cmds/verify_storage.go:121 > manifests checked␣
↪heights=[-1,1] module=command-database-verify
2021-06-08T10:56:21.210456408Z INF ../../../../pkg/mod/github.com/spikeekips/mitum@v0.0.
↪0-20210605063447-f720096b150d/launch/cmds/database_verify.go:105 > database verified␣
↪module=command-database-verify
```

2.12.4 clean

The `clean` command is used for cleaning blockdata and database.

```
$ ./mc storage clean <node design file>
```

EXAMPLE

```
$ ./mc storage clean node.yml
```

2.12.5 clean-by-height

The `clean-by-height` command is used for cleaning blockdata and database above a specific height.

```
$ ./mc storage clean-by-height <node design file> <height>
```

EXAMPLE

```
$ ./mc storage clean-by-height node.yml 54234
```

2.12.6 restore

The `restore` command is used for restoring the entire database from the downloaded blockdata.

When you use the `restore` command, both blockdata and data used for digest API are created. Check if the `network id` in the settings of the yml file is the same as the `network id` of the downloaded node.

- Multiple blockdata can be recovered simultaneously with the `-concurrency` option.
- If you want to delete and restore the existing mongodb data, use `-clean`.
- Use `-dryrun` to only check blockdata without actually recovering it.
- If you specify a specific blockdata directory with the `-one` option, you can recover them one by one.

```
$ ./mc storage restore <node design file>
```

EXAMPLE

```
$ ./mc storage restore node.yml --concurrency 10
2021-06-08T11:00:34.304594Z INF prepare to run module=command-restore
2021-06-08T11:00:34.304656Z INF prepared module=command-restore
2021-06-08T11:00:34.743477729Z INF block restored height=-1 module=command-restore
2021-06-08T11:00:34.828859729Z INF block restored height=0 module=command-restore
2021-06-08T11:00:34.829060729Z INF restored module=command-restore
2021-06-08T11:00:35.833206729Z INF stopped module=command-restore
```

2.12.7 set-blockdatamaps

The `set-blockdatamaps` command is used for updating multiple BlockDataMaps.

See *Block Data* for details.

```
$ ./mc storage set-blockdatamaps <deploy key> <maps file> [<node url>]
```

2.13 Deploy Command

Execute the `deploy key` command to create and manage the node's deploy key.

The subcommands related to `deploy key` command are as follows.

- `new`
- `keys`
- `key`
- `revoke`

Note: What is deploy key?

Updates of nodes (such as changing the BlockDataMap) should be allowed only by the node owner. The node owner uses the key to prove himself when managing the node.

However, it is dangerous to directly use a node's private key for node management. Thus, we need a **replaceable** and **manageable** key that can be used for things like node management.

`deploy key` is used for this purpose.

2.13.1 new

The `new` command is used for creating and registering a new deploy key to the node.

```
$ ./mc deploy key new <private key of node> <network-id> [<node url>]
```

EXAMPLE

```
$ NODE_PRV_KEY=KxaTHDAQnmFeWWik5MqWXBYkhvp5EpWbsZzXeHDdTDb5NE1dVw8wmpr

$ NODE=https://127.0.0.1:54321

$ NETWORK_ID=mitum

$ ./mc deploy key new $NODE_PRV_KEY $NETWORK_ID $NODE --tls-insecure
{"key":"d-fc4179e7-2ff3-4372-bd83-f70526bed476","added_at":"2021-06-09T09:31:22.
↪3216758522"}
2021-06-09T09:31:22.320055Z INF new deploy key module=command-deploy-key-new
```

2.13.2 keys

The keys command is used for obtaining the list of registered deploy keys in the node.

```
$ ./mc deploy key keys <private key of node> <network-id> [<node url>]
```

EXAMPLE

```
$ NODE_PRV_KEY=KxaTHDAQnmFeWWik5MqWXBYkhvp5EpWbsZzXeHDdTDb5NE1dVw8wmpr

$ NODE=https://127.0.0.1:54321

$ NETWORK_ID=mitum

$ ./mc deploy key keys $NODE_PRV_KEY $NETWORK_ID $NODE --tls-insecure
[{"key":"d-974702df-89a7-4fd1-a742-2d66c1ead6cd","added_at":"2021-06-09T03:14:33.9Z"},{
↪"key":"d-2897ced4-ceb5-4e11-be81-3139350c9c55","added_at":"2021-06-09T03:56:49.393Z"},{
↪"key":"d-fc4179e7-2ff3-4372-bd83-f70526bed476","added_at":"2021-06-09T09:31:22.
↪321675852Z"}]
```

2.13.3 key

The key command is used for checking the existence of the deploy key in the node.

```
$ ./mc deploy key key <deploy key> <private key of node> <network-id> [<node url>]
```

EXAMPLE

```
$ NODE_PRV_KEY=KxaTHDAQnmFeWWik5MqWXBYkhvp5EpWbsZzXeHDdTDb5NE1dVw8wmpr

$ NODE=https://127.0.0.1:54321

$ NETWORK_ID=mitum

$ DEPLOY_KEY=d-974702df-89a7-4fd1-a742-2d66c1ead6cd

$ ./mc deploy key key $DEPLOY_KEY $NODE_PRV_KEY $NETWORK_ID $NODE --tls-insecure
{"key":"d-974702df-89a7-4fd1-a742-2d66c1ead6cd","added_at":"2021-06-09T03:14:33.9Z"}
```

2.13.4 revoke

The `revoke` command is used for revoking the deploy key from the node.

```
$ ./mc deploy key revoke <deploy key> <private key of node> <network-id> [<node url>]
```

EXAMPLE

```
$ NODE_PRV_KEY=KxaTHDAQnmFeWWik5MqWXBYkhvp5EpWbsZzXeHDdTDb5NE1dVw8wmpr

$ NODE=https://127.0.0.1:54321

$ NETWORK_ID=mitum

$ DEPLOY_KEY=d-974702df-89a7-4fd1-a742-2d66c1ead6cd

$ ./mc deploy key revoke $DEPLOY_KEY $NODE_PRV_KEY $NETWORK_ID $NODE --tls-insecure
2021-06-09T09:36:19.763339Z INF deploy key revoked deploy_key=d-974702df-89a7-4fd1-a742-
↪2d66c1ead6cd module=command-deploy-key-revoke
```

2.14 Others

- `version`
- `quic-client`

2.14.1 version

Check the version of the installed Mitum Currency using the `version` command.

```
$ ./mc version
```

EXAMPLE

```
$ ./mc version
v0.0.1
```

2.14.2 quic-client

Note that the response of `quic-client` is identical to the response when requesting *node info* by API.

```
$ ./mc quic-client <node-url>
```

EXAMPLE

```
$ ./mc quic-client https://3.35.171.179:54321/
{
    "_hint": "node-info-v0.0.1",
    "node": {
        "_hint": "base-node-v0.0.1",
        "address": "node4sas",
        "publickey": "21im86HvT3aC4p23AExN7PKRD3RF1GR8cD3E95iEJHhNKmpu"
    },
    "network_id": "bWl0dW0=",
    "state": "CONSENSUS",
    "last_block": {
        "_hint": "block-manifest-v0.0.1",
        "hash": "GBQqKbR6pAs8gWzNmf5mrHGUYUmjs829NVX4WuYz7uzf",
        "height": 994024,
        "round": 0,
        "proposal": "HbxL38mNX8NGTqErNE3Hw5w639qKpbEwC4SkkCDZvrYB",
        "previous_block": "5rPQHEunbAw15YG3GaZneYKQpxsKRgQuThW6Yd7KBZb",
        "block_operations": null,
        "block_states": null,
        "confirmed_at": "2022-01-19T05:58:14.623577286Z",
        "created_at": "2022-01-19T05:58:14.631963244Z"
    },
    "version": "v0.0.1-stable-383cf0c-20211224",
    "policy": {
        "timespan_valid_ballot": 60000000000,
        "network_connection_timeout": 3000000000,
        "threshold": 100,
        "max_operations_in_seal": 10,
        "max_operations_in_proposal": 100,
        "interval_broadcasting_proposal": 1000000000,
        "wait_broadcasting_accept_ballot": 1000000000,
        "timeout_waiting_proposal": 5000000000,
        "interval_broadcasting_init_ballot": 1000000000,
        "interval_broadcasting_accept_ballot": 1000000000,
        "suffrage": "{\"type\":\"\",\"cache_size\":10,\"number_of_acting\":1}"
    },
    "suffrage": [
        {
            "address": "node4sas",
            "publickey": "21im86HvT3aC4p23AExN7PKRD3RF1GR8cD3E95iEJHhNKmpu",
            "conninfo": {
```

(continues on next page)

(continued from previous page)

```
                "_hint": "http-conninfo-v0.0.1",
                "url": "https://3.35.171.179:54321",
                "insecure": true
            }
        }
    ],
    "conninfo": {
        "_hint": "http-conninfo-v0.0.1",
        "url": "https://3.35.171.179:54321",
        "insecure": true
    }
}
```

2.15 Lookup Account

2.15.1 Prerequisite

- curl
  - This is a command line tool for interacting with API.
  - https://curl.se
- jq
  - This is a command line tool for parsing json response.
  - https://stedolan.github.io/jq/

2.15.2 Genesis Account Lookup

1. You can look up genesis account from local blockdata.

```
$ AC0_ACC_KEY=GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623-mca:account

$ find blockfs -name "*-states-*" -print | xargs -n 1 gzcat | grep '^{' | jq '. |␣
↪select(.key == "'$AC0_ACC_KEY'") | [ "height: "+(.height|tostring), "state_key: " + .
↪key, "address: " + .value.value.address, .operations, .value.value.keys.keys, .value.
↪value.keys.threshold]'
[
    "height: 2",
    "state_key: GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623-mca:account",
    "address: FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
    [
        "9mc8YEFWC27WEF3VVee1wk4ib5kvWBk1iJ41pWf27Mrc"
    ],
    [
        {
            "_hint": "mitum-currency-key-v0.0.1",
            "weight": 100,
            "key": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
```

(continues on next page)

(continued from previous page)

```
        }
    ],
    100
]
```

- 99999999999999999977 = 99999999999999999999 - (2 create account: 10 * 2) - (2 fee: 1 * 2)

2. You can also look up genesis account from digest api.

```
$ curl --insecure https://localhost:54320/account/
↪FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca | jq '{_embedded}'
{
    "_embedded": {
        "_hint": "mitum-currency-account-value-v0.0.1",
        "hash": "AHQ4ohzwm7Y9T3f8vH5LTQ2rXKfVg3eazCdyihqsWv8F",
        "address": "FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca",
        "keys": {
            "_hint": "mitum-currency-keys-v0.0.1",
            "hash": "GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623",
            "keys": [
                {
                    "_hint": "mitum-currency-key-v0.0.1",
                    "weight": 100,
                    "key": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                }
            ],
            "threshold": 100
        },
        "balance": [
            {
                "_hint": "mitum-currency-amount-v0.0.1",
                "amount": "9999999999999",
                "currency": "MCC2"
            },
            {
                "_hint": "mitum-currency-amount-v0.0.1",
                "amount": "50",
                "currency": "MCC"
            }
        ],
        "height": 10,
        "previous_height": -2
    }
}
```

Note: When you look up state by address from mongodb, remove the part after – of the address and use it as a key.

- `FnuHC5HkFMpr4QABukchEeT63612gGKus3cRK3KAqK7Bmca` → `GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623-mca`

2.16 REST API

Digest API is a service that allows nodes to search blockchain data.
It can be used in various applications, such as wallet or blockchain explorer.

- API is provided through *HTTP/2 network protocol*.
- Response message follows HAL and is delivered in *JSON* format.
- API data storage can be set in *Configuration* of Mitum Currency.
- Mitum's main storage can be used, or a separate database is also possible.
- *TLS certificates* required for *HTTP/2* will randomly generate self signed certificates if the service host is localhost unless the path of the file is set separately.

If an operation is not included in the block due to a specific problem, the cause can be checked through the response.

2.16.1 Summary

This is the summary of REST API for Mitum Currency.

REQUEST URL	METHOD	RESPONSE
/	GET	Node information
/block/manifests	GET	All block manifests
/block/{height}	GET	Block by block height
/block/{height}/manifest	GET	Block manifest by block height
/block/{height}/operations	GET	All operations of block
/block/{block_hash}	GET	Block by block hash
/block/{block_hash}/manifest	GET	Block manifest by block hash
/block/operations	GET	All operations
/block/operation/{fact_hash}	GET	Operation by fact hash
/account/{address}	GET	Latest state of account
/account/{address}/operations	GET	Operations related to account
/accounts?publickey={public_key}	GET	Accounts related to public key
/builder/operation	GET	Available Operation types
/builder/operation/fact/template/{fact}	GET	Fact template
/builder/operation/fact	POST {fact}	Operation message from fact
/builder/operation/sign	POST {operation}	Operation with hash from operation
/builder/send	POST {operation/seal}	Broadcast seal
/currency	GET	All currencies
/currency/{currency_id}	GET	Currency by currency id

Refer to Mitum Currency Digest API Docs for details.

This document doesn't provide any information of APIs for Mitum Blocksign or Mitum Blockcity. See Mitum Blocksign Digest API Docs for Mitum Blocksign.

2.17 Using Operation Builder

Digest API has **Operation Builder** to help write operation messages. **Operation Builder** makes it possible to generate operation messages through api without using SDK.

2.17.1 Get Operation Fact Template

By requesting an *Operation Fact Template*, you can receive a template for each operation type. You can create a fact message by changing the field value of the template.

METHOD	GET
PATH	/builder/operation/fact/template/{fact_type}

RESPONSE EXAMPLE

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
        "hash": "3Zdg5ZVdNFRbwX5WU7Nada3Wnx5VEgkHrDLVLkE8FMs1",
        "token": "cmFpc2VkIGJ5",
        "sender": "mothermca",
        "items": [
            {
                "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                "keys": {
                    "_hint": "mitum-currency-keys-v0.0.1",
                    "hash": "2TQ8Xn5tdowqkJt8kHWcNj2QKhuNRnnCwiXxFbRbwBWY",
                    "keys": [
                        {
                            _hint": "mitum-currency-key-v0.0.1",
                            "weight": 100,
                            "key": "oRHdEPPrgbfNxUp6TWsC35DmWu1zbLCW9rp41Z8npF8Hmpu"
                        }
                    ],
                    "threshold": 100
                },
                "amounts": [
                    {
                        "_hint": "mitum-currency-amount-v0.0.1",
                        "amount": "-333",
                        "currency": "xXx"
                    }
                ]
            }
        ]
```

(continues on next page)

(continued from previous page)

```
    },
    "_links": {
        "self": {
            "href": "/builder/operation/fact/template/create-accounts"
        }
    },
    "_extra": {
        "default": {
            "items.keys.keys.key": "oRHdEPPrgbfNxUp6TWsC35DmWu1zbLCW9rp41Z8npF8Hmpu"
            "items.big": "-333",
            "currency": "xXx",
            "token": "cmFpc2VkIGJ5",
            "sender": "mothermca",
        }
    }
}
```

- The _embedded object among the contents of the template responded represents the *fact*. Edit the contents of the fact json object and use it in *Build Operation Message*.

create-accounts FACT EXAMPLE

```
{
    "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
    "hash": "3Zdg5ZVdNFRbwX5WU7Nada3Wnx5VEgkHrDLVLkE8FMs1",
    "token": "cmFpc2VkIGJ5",
    "sender": "mothermca",
    "items": [
        {
            "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
            "keys": {
                "_hint": "mitum-currency-keys-v0.0.1",
                "hash": "2TQ8Xn5tdowqkJt8kHWcNj2QKhuNRnnCwiXxFbRbwBWY",
                "keys": [
                    {
                        _hint": "mitum-currency-key-v0.0.1",
                        "weight": 100,
                        "key": "oRHdEPPrgbfNxUp6TWsC35DmWu1zbLCW9rp41Z8npF8Hmpu"
                    }
                ],
                "threshold": 100
            },
            "amounts": [
                {
                    "_hint": "mitum-currency-amount-v0.0.1",
                    "amount": "-333",
                    "currency": "xXx"
                }
            ]
        }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

- There is no need to edit the `hash` value as the builder automatically completes it.
- `token` is a *base64* encoded value.
- Use the `_hint` item as it is.

Check *Key Command* for the details of key registration of accounts related to `keys`.

2.17.2 Build Operation Message

The created fact message is sent to the request body in json format and the completed fact message is received.

In the case of the example, you will receive a fact message with the `keys hash`, `token`, and `fact hash` changed.

METHOD	POST
PATH	/builder/operation/fact

RESPONSE EXAMPLE

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-create-accounts-operation-v0.0.1",
    "_embedded": {
        "hash": "92FXbSdm46iuA7kQuC6ENfi5pd64G1Uiu49A3VmaA8Tu",
        "fact": {
            "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
            "hash": "9ttqrz1bkFNCySVnrhYrxewcVB6mkZWWvBpSPS2fShip",
            "token": "MjAyMS0wNi0xNSAwODo0OTozOS45NDggKzAwMDAgVVRD",
            "sender": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
            "items": [
                {
                    "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                    "keys": {
                        "_hint": "mitum-currency-keys-v0.0.1",
                        "hash": "GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623",
                        "keys": [
                            {
                                "_hint": "mitum-currency-key-v0.0.1",
                                "weight": 100,
                                "key": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                            }
                        ],
```

(continues on next page)

(continued from previous page)

```
                        "threshold": 100
                    },
                    "amounts": [
                        {
                            "_hint": "mitum-currency-amount-v0.0.1",
                            "amount": "333",
                            "currency": "MCC"
                        }
                    ]
                }
            ]
        },
        "fact_signs": [
            {
                "_hint": "base-fact-sign-v0.0.1",
                "signer": "oRHdEPPrgbfNxUp6TWsC35DmWu1zbLCW9rp41Z8npF8Hmpu",
                "signature": "22UZo26eN",
                "signed_at": "2020-10-08T07:53:26Z"
            }
        ],
        "memo": "",
        "_hint": "mitum-currency-create-accounts-operation-v0.0.1"
    },
    "_links": {
        "self": {
            "href": "/builder/operation/fact"
        }
    },
    "_extra": {
        "default": {
            "fact_signs.signer": "oRHdEPPrgbfNxUp6TWsC35DmWu1zbLCW9rp41Z8npF8Hmpu",
            "fact_signs.signature": "22UZo26eN"
        },
        "signature_base": "hCi8MFOChFusqKx6v0zrsJ8u3tppYUOewadYjwTvDUFtaXR1bQ=="
    }
}
```

Check the `fact.hash` value of the response data. `fact.hash` value is used as data to complete the value of the fact_sign object.

In a *fact_sign* in `fact_signs`,

- The `signer` is the *publickey* of the keypair used to create the signature.
- The `signature` is generated by the `signer`.
- `signed_at` is the datetime at which the signature was generated.

2.17.3 Sign Operation Message

A *signature* is created using the hash of the received *fact* then the *fact_sign* for it is added.

When the generated fact message is sent to the request body in json format, the completed operation message with the *operation hash* added is received.

METHOD	POST
PATH	/builder/operation/sign

REQUEST BODY EXAMPLE

```
{
    "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
    "fact": {
        "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
        "hash": "CDUkHDJB4aC8552QvVCAPk8ZtohSuow67cPZZxqZG7RE",
        "token": "MjAyMS0wMy0yNCAwMjozNzozNC4xNzQgKzAwMDAgVVRD",
        "sender": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
        "items": [
            {
                "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                "keys": {
                    "_hint": "mitum-currency-keys-v0.0.1",
                    "hash": "GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623",
                    "keys": [
                        {
                            "_hint": "mitum-currency-key-v0.0.1",
                            "weight": 100,
                            "key": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                        }
                    ],
                    "threshold": 100
                },
                "amounts": [
                    {
                        "_hint": "mitum-currency-amount-v0.0.1",
                        "amount": "333",
                        "currency": "MCC"
                    }
                ]
            }
        ]
    },
    "fact_signs": [
        {
            "_hint": "base-fact-sign-v0.0.1",
            "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
```

(continues on next page)

(continued from previous page)

```
            "signature":
↪"AN1rKvtVhunuSdS8g3KWQ1PFBEP9bzz4sU4Vb3B4JrYyVUF79XwNUrG6AzoVfq6mHsK8W4S5hu7LKjDARfAQeDWwit1GnKXcN
↪",
            "signed_at": "2021-06-16T01:56:14.124268Z"
        }
    ],
    "memo": "",
}
```

RESPONSE EXAMPLE

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-create-accounts-operation-v0.0.1",
    "_embedded": {
        "fact": {
            "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
            "hash": "CDUkHDJB4aC8552QvVCAPk8ZtohSuow67cPZZxqZG7RE",
            "token": "MjAyMS0wMy0yNCAwMjozNzozNC4xNzQgKzAwMDAgVVRD",
            "sender": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
            "items": [
                {
                    "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                    "keys": {
                        "_hint": "mitum-currency-keys-v0.0.1",
                        "hash": "GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623",
                        "keys": [
                            {
                                "_hint": "mitum-currency-key-v0.0.1",
                                "weight": 100,
                                "key": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                            }
                        ],
                        "threshold": 100
                    },
                    "amounts": [
                        {
                            "_hint": "mitum-currency-amount-v0.0.1",
                            "amount": "333",
                            "currency": "MCC"
                        }
                    ]
                }
            ]
        },
        "fact_signs": [
            {
                "_hint": "base-fact-sign-v0.0.1",
                "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
```

(continues on next page)

(continued from previous page)

```
                "signature":
↪"AN1rKvtVhunuSdS8g3KWQ1PFBEP9bzz4sU4Vb3B4JrYyVUF79XwNUrG6AzoVfq6mHsK8W4S5hu7LKjDARfAQeDWwit1GnKXcN
↪",
                "signed_at": "2021-06-16T01:56:14.124268Z"
            }
        ],
        "memo": "",
        "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
        "hash": "9pNsg6gkQJoVsB7iqY3udeLVti2Yxgbe4mFkGqzds2AT"
    },
    "_links": {
        "self": {
            "href": "/builder/operation/sign"
        }
    }
}
```

2.17.4 Broadcast Message to Network

By requesting an *Operation* or *Seal* message as the request body, you can broadcast it to the network.

In this case, the *signer* of the seal becomes the digest node.

- If the request body is an **operation**, a new seal is created and the digest node signs.
- If the request body is a **seal**, the seal is signed by the digest node.

METHOD	POST
PATH	/builder/send

REQUEST BODY EXAMPLE

```
{
    "fact": {
        "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
        "hash": "CDUkHDJB4aC8552QvVCAPk8ZtohSuow67cPZZxqZG7RE",
        "token": "MjAyMS0wMy0yNCAwMjozNzozNC4xNzQgKzAwMDAgVVRD",
        "sender": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
        "items": [
            {
                "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                "keys": {
                    "_hint": "mitum-currency-keys-v0.0.1",
                    "hash": "GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623",
                    "keys": [
                        {
                            "_hint": "mitum-currency-key-v0.0.1",
```

(continues on next page)

(continued from previous page)

```
                            "weight": 100,
                            "key": "2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                        }
                    ],
                    "threshold": 100
                },
                "amounts": [
                    {
                        "_hint": "mitum-currency-amount-v0.0.1",
                        "amount": "333",
                        "currency": "MCC"
                    }
                ]
            }
        ]
    },
    "fact_signs": [
        {
            "_hint": "base-fact-sign-v0.0.1",
            "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
            "signature":
↪"AN1rKvtVhunuSdS8g3KWQ1PFBEP9bzz4sU4Vb3B4JrYyVUF79XwNUrG6AzoVfq6mHsK8W4S5hu7LKjDARfAQeDWwit1GnKXcN
↪",
            "signed_at": "2021-06-16T01:56:14.124268Z"
        }
    ],
    "memo": "",
    "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
    "hash": "9pNsg6gkQJoVsB7iqY3udeLVti2Yxgbe4mFkGqzds2AT"
}
```

RESPONSE EXAMPLE

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "seal-v0.0.1",
    "_embedded": {
        "_hint": "seal-v0.0.1",
        "hash": "4UvusVw9RYdqxHQz2EzDb6gW6CgoZGPayD1yZBcdSSHW",
        "body_hash": "9AFx2gAqeMveV6ojwUi6HKx19GfbZZggPTGhTS3dDih5",
        "signer": "uGnKHNfh8EtNVXsL4Qu1a655oQuzibK8Tc41TZUHzHqkmpu",
        "signature":
↪"381yXZAzT6LcYUXfTG9Fifc6neDfXDqpjzuGzfqr1LXPMvvtseJKzGSRwdL6jvkHBaVRdGPD4YfrHnp2rbpZEEWRNAePiJBt
↪",
        "signed_at": "2021-06-16T03:06:33.649190888Z",
        "operations": [
            {
                "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
                "hash": "9pNsg6gkQJoVsB7iqY3udeLVti2Yxgbe4mFkGqzds2AT",
```

(continues on next page)

(continued from previous page)

```
                "fact": {
                    "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                    "hash": "CDUkHDJB4aC8552QvVCAPk8ZtohSuow67cPZZxqZG7RE",
                    "token": "MjAyMS0wMy0yNCAwMjozNzozNC4xNzQgKzAwMDAgVVRD",
                    "sender": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
                    "items": [
                        {
                            "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1
↪",
                            "keys": {
                                "_hint": "mitum-currency-keys-v0.0.1",
                                "hash": "GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623",
                                "keys": [
                                    {
                                        "_hint": "mitum-currency-key-v0.0.1",
                                        "weight": 100,
                                        "key":
↪"2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                                    }
                                ],
                                "threshold": 100
                            },
                            "amounts": [
                                {
                                    "_hint": "mitum-currency-amount-v0.0.1",
                                    "amount": "333",
                                    "currency": "MCC"
                                }
                            ]
                        }
                    ]
                },
                "fact_signs": [
                    {
                        "_hint": "base-fact-sign-v0.0.1",
                        "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
                        "signature":
↪"AN1rKvtVhunuSdS8g3KWQ1PFBEP9bzz4sU4Vb3B4JrYyVUF79XwNUrG6AzoVfq6mHsK8W4S5hu7LKjDARfAQeDWwit1GnKXcN
↪",
                        "signed_at": "2021-06-16T01:56:14.124268Z"
                    }
                ],
                "memo": ""
            }
        ]
    },
    "_links": {
        "self": {
            "href": ""
        },
        "operation:0": {
            "href": "/block/operation/CDUkHDJB4aC8552QvVCAPk8ZtohSuow67cPZZxqZG7RE"
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

2.17.5 Confirming the Success of the Operation

By querying the operation with the *fact hash* value in the api, you can check whether the operation is successfully processed or not.

METHOD	GET
PATH	/block/operation/{operation_fact_hash}

- If `_embedded.in_state` is true in the response message, it means the operation has successfully been saved in the block.
- If `_embedded.in_state` is false, it means the operation hasn't been saved in the block.
- If **the operation fails**, the reason may be as follows.
  1. *insufficient balance of sender* when sending money
  2. *incorrect signature*
  3. *create-account amount less than new-account-min-balance*
  4. etc…

You can check the reason for failure in `_embedded.reason.msg` in the response message.

RESPONSE EXAMPLE

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-operation-value-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-operation-value-v0.0.1",
        "hash": "CDUkHDJB4aC8552QvVCAPk8ZtohSuow67cPZZxqZG7RE",
        "operation": {
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "9pNsg6gkQJoVsB7iqY3udeLVti2Yxgbe4mFkGqzds2AT",
            "fact": {
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "CDUkHDJB4aC8552QvVCAPk8ZtohSuow67cPZZxqZG7RE",
                "token": "MjAyMS0wMy0yNCAwMjozNzozNC4xNzQgKzAwMDAgVVRD",
                "sender": "CoXPgSxcad3fRAbp2JBEeGcYGEQ7dQhdZGWXLbTHpwuGmca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                        "keys": {
```

(continues on next page)

(continued from previous page)

```
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "GkswusUGC22R5wmrXWB5yqFm8UN22yHLihZMkMb3z623",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"2Aopgs1nSzNCWLvQx5fkBJCi2uxjYBfN8TqneqFd9DzGcmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "333",
                                "currency": "MCC"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                    "_hint": "base-fact-sign-v0.0.1",
                    "signer": "rcrd3KA2wWNhKdAP8rHRzfRmgp91oR9mqopckyXRmCvGmpu",
                    "signature":
↪"AN1rKvtVhunuSdS8g3KWQ1PFBEP9bzz4sU4Vb3B4JrYyVUF79XwNUrG6AzoVfq6mHsK8W4S5hu7LKjDARfAQeDWwit1GnKXcN
↪",
                    "signed_at": "2021-06-16T01:56:14.124Z"
                }
            ],
            "memo": ""
        },
        "height": 108674,
        "confirmed_at": "2021-06-16T02:26:55.75Z",
        "reason": {
            "_hint": "base-operation-reason-v0.0.1",
            "msg": "state, \"9g4BAB8nZdzWmrsAomwdvNJU2hA2psvkfTQ5XdLn4F4r-mca:account\"␣
↪does not exist",
            "data": null
        },
        "in_state": false,
        "index": 0
    },
    "_links": {
        "manifest": {
            "href": "/block/108674/manifest"
        },
        "operation:{hash}": {
            "templated": true,
            "href": "/block/operation/{hash:(?i)[0-9a-z][0-9a-z]+}"
```

(continues on next page)

(continued from previous page)

```
        },
        "block:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "self": {
            "href": "/block/operation/CDUkHDJB4aC8552QvVCAPk8ZtohSuow67cPZZxqZG7RE"
        },
        "block": {
            "href": "/block/108674"
        }
    }
}
```

2.18 API List

This is the page of the explanation for each API Path.

For details, visit Mitum Currency Digest API Docs.

2.18.1 Node Info

/

- It returns the node information of the network.

PATH	METHOD
/	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "node-info-v0.0.1",
    "_embedded": {
        "_hint": "node-info-v0.0.1",
        "node": {
            "_hint": "base-node-v0.0.1",
            "address": "node4sas",
            "publickey": "21im86HvT3aC4p23AExN7PKRD3RF1GR8cD3E95iEJHhNKmpu"
        },
        "network_id": "bWl0dW0=",
        "state": "CONSENSUS",
```

(continues on next page)

(continued from previous page)

```
        "last_block": {
            "_hint": "block-manifest-v0.0.1",
            "hash": "7KjDLJMMzKw6RtfwjoZZ75rcab15mn6ASjQbGjotX1NW",
            "height": 1622504,
            "round": 0,
            "proposal": "H9ztzWj46ayufSvBXjdNXo7Xs3q2DK8nj9exaMPo1iyt",
            "previous_block": "JeT7t26J279p3EWq1S1yAEdXUqu4EYZa9tprT9nkMCy",
            "block_operations": null,
            "block_states": null,
            "confirmed_at": "2022-02-03T04:47:01.355363841Z",
            "created_at": "2022-02-03T04:47:01.361237906Z"
        },
        "version": "v0.0.1-stable-383cf0c-20211224",
        "policy": {
            "interval_broadcasting_accept_ballot": 1000000000,
            "timespan_valid_ballot": 60000000000,
            "max_operations_in_seal": 10,
            "interval_broadcasting_proposal": 1000000000,
            "wait_broadcasting_accept_ballot": 1000000000,
            "interval_broadcasting_init_ballot": 1000000000,
            "network_connection_timeout": 3000000000,
            "suffrage": "{\"cache_size\":10,\"number_of_acting\":1,\"type\":\"\"}",
            "threshold": 100,
            "max_operations_in_proposal": 100,
            "timeout_waiting_proposal": 5000000000
        },
        "suffrage": [
            {
                "address": "node4sas",
                "publickey": "21im86HvT3aC4p23AExN7PKRD3RF1GR8cD3E95iEJHhNKmpu",
                "conninfo": {
                    "_hint": "http-conninfo-v0.0.1",
                    "url": "https://3.35.171.179:54321",
                    "insecure": true
                }
            }
        ],
        "conninfo": {
            "_hint": "http-conninfo-v0.0.1",
            "url": "https://3.35.171.179:54321",
            "insecure": true
        }
    },
    "_links": {
        "block:{hash}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "currency:{currencyid}": {
            "templated": true,
            "href": "/currency/{currencyid:.*}"
        },
```

(continues on next page)

(continued from previous page)

```
        "block:current": {
            "href": "/block/1622504"
        },
        "block:current-manifest": {
            "href": "/block/1622504/manifest"
        },
        "block:manifest:{hash}": {
            "templated": true,
            "href": "/block/{hash:(?i)[0-9a-z][0-9a-z]+}/manifest"
        },
        "block:next": {
            "href": "/block/1622505"
        },
        "block:prev": {
            "href": "/block/1622503"
        },
        "self": {
            "href": "/"
        },
        "currency": {
            "href": "/currency"
        },
        "block:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "block:manifest:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}/manifest"
        }
    }
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

2.18.2 Block

/block/manifests

- It returns all block manifests of the network.

PATH	METHOD
/block/manifests	GET

Query		Example
offset	manifests after offset - block height	2
reverse	manifests by reverse order	1 (true)

- offset: integer; block height
- reverse: boolean; use 1 for `true`

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "",
    "_embedded": [
        {
            "_hint": "mitum-currency-hal-v0.0.1",
            "hint": "block-manifest-v0.0.1",
            "_embedded": {
                "_hint": "block-manifest-v0.0.1",
                "hash": "F3qqMUDjofiLkftSSHn4N6uZYBppQzc48iKs8Aqupe9b",
                "height": 1,
                "round": 0,
                "proposal": "34VNRjGW3TqrQ455dyqoKp1EDUeAvu3VfnyLu3aZDcur",
                "previous_block": "6AMoeUTpDfF2Vs73HRWRCVfkqnVLs6gSwUpXYbJzDmAV",
                "block_operations": null,
                "block_states": null,
                "confirmed_at": "2021-12-26T04:22:10.627Z",
                "created_at": "2021-12-26T04:22:10.639Z"
            },
            "_links": {
                "next": {
                    "href": "/block/2/manifest"
                },
                "block": {
                    "href": "/block/1"
                },
                "block:{height}": {
                    "templated": true,
                    "href": "/block/{height:[0-9]+}"
                },
```

(continues on next page)

(continued from previous page)

```
                "block:{hash}": {
                    "href": "/block/{height:[0-9]+}",
                    "templated": true
                },
                "manifest:{height}": {
                    "templated": true,
                    "href": "/block/{height:[0-9]+}/manifest"
                },
                "manifest:{hash}": {
                    "templated": true,
                    "href": "/block/{hash:(?i)[0-9a-z][0-9a-z]+}/manifest"
                },
                "self": {
                    "href": "/block/1/manifest"
                },
                "alternate": {
                    "href": "/block/F3qqMUDjofiLkftSSHn4N6uZYBppQzc48iKs8Aqupe9b/manifest
↪"
                }
            }
        },
        ...
    ],
    "_links": {
        "next": {
        "href": "/block/manifests?offset=10"
        },
        "reverse": {
        "href": "/block/manifests?reverse=1"
        },
        "self": {
        "href": "/block/manifests?offset=0"
        }
    }
}
```

- 404 (No more manifests)

If there are no more manifests, it returns `404`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "manifests not found",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/block/{height}

- It returns the block information of the block by *block height*.

PATH	METHOD
/block/{height}	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "",
    "_links": {
        "self": {
            "href": "/block/5"
        },
        "manifest:{hash}": {
            "templated": true,
            "href": "/block/{hash:(?i)[0-9a-z][0-9a-z]+}/manifest"
        },
        "prev": {
            "href": "/block/4"
        },
        "current": {
            "href": "/block/5"
        },
        "current-manifest": {
            "href": "/block/5/manifest"
        },
        "block:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "block:{hash}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "manifest:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}/manifest"
```

(continues on next page)

(continued from previous page)

```
        },
        "next": {
            "href": "/block/6"
        }
    }
}
```

- 400 (block not found)

If the height you request with is wrong, it returns `400`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "title": "bad request; invalid height found for block by height: strconv.ParseInt:␣
↪parsing \"...\": value out of range",
    "detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/block/{height}/manifest

- It returns the block manifest of the block by *block height*.

PATH	METHOD
/block/{height}/manifest	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "block-manifest-v0.0.1",
    "_embedded": {
        "_hint": "block-manifest-v0.0.1",
        "hash": "9zVqaLhLngT8gmTUXfRNLo7WxGQBYoZkYw4NSDrKTrvX",
        "height": 222,
        "round": 0,
```

(continues on next page)

(continued from previous page)

```
        "proposal": "66yixQwnHwBaJ4qDfpwsTa2tBgDGXYHGT1Nta7jD24S1",
        "previous_block": "CyPXbZUAhRb5dH2cJHJtfw51H73NwLSkyz1Ad7iWrpDc",
        "block_operations": null,
        "block_states": null,
        "confirmed_at": "2021-12-26T04:29:44.869Z",
        "created_at": "2021-12-26T04:29:44.877Z"
    },
    "_links": {
        "alternate": {
            "href": "/block/9zVqaLhLngT8gmTUXfRNLo7WxGQBYoZkYw4NSDrKTrvX/manifest"
        },
        "next": {
            "href": "/block/223/manifest"
        },
        "block": {
            "href": "/block/222"
        },
        "block:{hash}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "manifest:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}/manifest"
        },
        "manifest:{hash}": {
            "templated": true,
            "href": "/block/{hash:(?i)[0-9a-z][0-9a-z]+}/manifest"
        },
        "block:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "self": {
            "href": "/block/222/manifest"
        }
    }
}
```

- 400 (manifest not found)

If the height you request with is wrong, it returns `400`.

```
{
    "title": "invalid height found for manifest by height",
    "detail": "...",
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others"
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/block/{height}/operations

- It returns all operations of the block by *block height*.

PATH	METHOD
/block/{height}/operations	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "",
    "_embedded": [
        {
            "_hint": "mitum-currency-hal-v0.0.1",
            "hint": "mitum-currency-operation-value-v0.0.1",
            "_embedded": {
                "_hint": "mitum-currency-operation-value-v0.0.1",
                "hash": "FXRvh8ovbAJdmwdz66gtgb1EJSAaSZkA5TadV8KD1oGs",
                "operation": {
                    "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
                    "hash": "EikTtWw8izGuaAWbu8dP7PRKpc5Ri6qYzPxaxaD7fr2r",
                    "fact": {
                        "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                        "hash": "FXRvh8ovbAJdmwdz66gtgb1EJSAaSZkA5TadV8KD1oGs",
                        "token": "MjAyMS0xMi0yN1QwNzo1ODo1My4zMDE3NjcrMDA6MDA=",
                        "sender": "5om5ZuSsqjEj7CxoF1VyLLJYhQoCwBPjUciy9gu8dh8hmca",
                        "items": [
                            {
                                "_hint": "mitum-currency-create-accounts-single-amount-
↪v0.0.1",
                                "keys": {
                                    "_hint": "mitum-currency-keys-v0.0.1",
                                    "hash": "C7ntk12BMkjpBoita2qf6USE45moRmLcrpUXn2FxCB31
↪",
                                    "keys": [
                                        {
                                            "_hint": "mitum-currency-key-v0.0.1",
                                            "weight": 100,
```

(continues on next page)

(continued from previous page)

```
                                              "key":
↪"2BfVL17JezsZjsYx3PzXW9aRzERFA4F2Hnj1bFK7akXhAmpu"
                                          }
                                      ],
                                      "threshold": 100
                                  },
                                  "amounts": [
                                      {
                                          "_hint": "mitum-currency-amount-v0.0.1",
                                          "amount": "100",
                                          "currency": "MCC"
                                      }
                                  ]
                              }
                          ]
                      },
                      "fact_signs": [
                          {
                              "_hint": "base-fact-sign-v0.0.1",
                              "signer": "p4nHuxamW5HQZQd1mpkMqsHCbnnwdjWZ9c21eF2eKdLrmpu",
                              "signature":
↪"AN1rKvtMWpbB3qLou12rkGVXxJxW4kYitEYkagNQJ4QWCYgNYSrLvsxDkLMxRfW2Do9KhkvzPVrr3r48YPN775yiJiMiyGx5m
↪",
                              "signed_at": "2021-12-27T07:58:53.323Z"
                          }
                      ],
                      "memo": ""
                  },
                  "height": 48480,
                  "confirmed_at": "2021-12-27T08:01:54.507Z",
                  "reason": {
                      "_hint": "base-operation-reason-v0.0.1",
                      "msg": "; state, \
↪"5om5ZuSsqjEj7CxoF1VyLLJYhQoCwBPjUciy9gu8dh8hmca:account\" does not exist",
                      "data": null
                  },
                  "in_state": false,
                  "index": 0
              },
              "_links": {
                  "block": {
                      "href": "/block/48480"
                  },
                  "manifest": {
                      "href": "/block/48480/manifest"
                  },
                  "self": {
                      "href": "/block/operation/
↪FXRvh8ovbAJdmwdz66gtgb1EJSAaSZkA5TadV8KD1oGs"
                  }
              }
          }
```

(continues on next page)

(continued from previous page)

```
    ],
    "_links": {
        "reverse": {
            "href": "/block/48480/operations?reverse=1"
        },
        "next": {
            "href": "/block/48480/operations?offset=0"
        },
        "self": {
            "href": "/block/48480/operations"
        }
    }
}
```

- 404 (operations not found)

If there are no more operations or there aren't any operations, it returns `404`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "operations not found",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/block/{block_hash}

- It returns the block information of the block by *block hash*.

PATH	METHOD
/block/{block_hash}	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
```

(continues on next page)

(continued from previous page)

```
    "hint": "",
    "_links": {
        "manifest": {
            "href": "/block/7tAfifVzxSz3kKzGq9RceKtuVAeFB7E9jvCUnojV3YfM/manifest"
        },
        "manifest:{height}": {
            "href": "/block/{height:[0-9]+}/manifest",
            "templated": true
        },
        "manifest:{hash}": {
            "templated": true,
            "href": "/block/{hash:(?i)[0-9a-z][0-9a-z]+}/manifest"
        },
        "block:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "block:{hash}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "self": {
            "href": "/block/7tAfifVzxSz3kKzGq9RceKtuVAeFB7E9jvCUnojV3YfM"
        }
    }
}
```

- 400 (block not found)

If the block hash is wrong, it returns `400`.

```
{
    "detail": "..."
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "title": "bad request; invalid hash for block by hash: invalid; empty hash"
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/block/{block_hash}/manifest

- It returns the block manifest of the block by *block hash*.

PATH	METHOD
/block/{block_hash}/manifest	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "block-manifest-v0.0.1",
    "_embedded": {
        "_hint": "block-manifest-v0.0.1",
        "hash": "7tAfifVzxSz3kKzGq9RceKtuVAeFB7E9jvCUnojV3YfM",
        "height": 1594489,
        "round": 0,
        "proposal": "3uSAcktWpnrB31RBKS35WABEMZDvDTEtvkMCDVLYfjR8",
        "previous_block": "3FiMUXRZTkcPQCcLwN2fhEP8C8xr9QrB4cK4yTentG59",
        "block_operations": "EF2cQGmrzW4AfUeZyEym7UCSbrMkXCcjhUadk2oM5ME2",
        "block_states": "YMyRynoNP11HfX9aMBjbw8bPWR1MVcaczqeLWV4wve8",
        "confirmed_at": "2022-02-02T12:53:44.669Z",
        "created_at": "2022-02-02T12:53:44.684Z"
    },
    "_links": {
        "block": {
            "href": "/block/1594489"
        },
        "manifest:{hash}": {
            "templated": true,
            "href": "/block/{hash:(?i)[0-9a-z][0-9a-z]+}/manifest"
        },
        "block:{height}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "block:{hash}": {
            "templated": true,
            "href": "/block/{height:[0-9]+}"
        },
        "manifest:{height}": {
            "href": "/block/{height:[0-9]+}/manifest",
            "templated": true
        },
        "self": {
            "href": "/block/1594489/manifest"
        },
        "alternate": {
            "href": "/block/7tAfifVzxSz3kKzGq9RceKtuVAeFB7E9jvCUnojV3YfM/manifest"
```

(continues on next page)

(continued from previous page)

```
        },
        "next": {
            "href": "/block/1594490/manifest"
        }
    }
}
```

- 404 (manifest not found)

If the block hash is wrong, it returns `404`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "title": "not found; manifest not found",
    "detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/block/operations

- It returns all operations of the network.

PATH	METHOD
/block/operations	GET

Query		Example
offset	manifests after offset - block height	2
reverse	manifests by reverse order	1 (true)

- offset: integer; block height
- reverse: boolean; use 1 for `true`

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "",
    "_embedded": [
        {
            "_hint": "mitum-currency-hal-v0.0.1",
            "hint": "mitum-currency-operation-value-v0.0.1",
            "_embedded": {
                "_hint": "mitum-currency-operation-value-v0.0.1",
                "hash": "7rSkwgF6BmLmid13jiBJKaaRtgYXS7rtDBFSuNdUNPeo",
                "operation": {
                    "_hint": "mitum-currency-genesis-currencies-operation-v0.0.1",
                    "hash": "2rtWNNHP15pBcdmmzCjsg45D5KPsqs49YPMRC8AtTJbo",
                    "fact": {
                        "_hint": "mitum-currency-genesis-currencies-operation-fact-v0.0.1
↪",
                        "hash": "7rSkwgF6BmLmid13jiBJKaaRtgYXS7rtDBFSuNdUNPeo",
                        "token": "bWl0dW0=",
                        "genesis_node_key":
↪"21im86HvT3aC4p23AExN7PKRD3RF1GR8cD3E95iEJHhNKmpu",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLu",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "currencies": [
                            {
                                "_hint": "mitum-currency-currency-design-v0.0.1",
                                "amount": {
                                    "_hint": "mitum-currency-amount-v0.0.1",
                                    "amount": "100000000000000000000000000000",
                                    "currency": "PEN"
                                },
                                "genesis_account": null,
                                "policy": {
                                    "_hint": "mitum-currency-currency-policy-v0.0.1",
                                    "new_account_min_balance": "10",
                                    "feeer": {
                                        "_hint": "mitum-currency-fixed-feeer-v0.0.1",
                                        "receiver":
↪"8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
                                        "amount": "1"
                                    }
                                },
                                "aggregate": "100000000000000000000000000000"
```

(continues on next page)

(continued from previous page)

```
                            },
                            ...
                        ]
                    },
                    "fact_signs": [
                        {
                            "_hint": "base-fact-sign-v0.0.1",
                            "signer": "21im86HvT3aC4p23AExN7PKRD3RF1GR8cD3E95iEJHhNKmpu",
                            "signature":
↪"AN1rKvt3e9wPJjbGEvucwxr7ntUX4oNBvsGmU4QQBFdAv1ToxXdCBqtbpJ7TwuqY1DyTCcS8FBQjJDbYsWpWixTTtXA5y3R5y
↪",
                            "signed_at": "2021-12-26T04:21:22.159Z"
                        }
                    ]
                },
                "height": 0,
                "confirmed_at": "2021-12-26T04:21:23.013Z",
                "reason": null,
                "in_state": true,
                "index": 0
            },
            "_links": {
                "block": {
                    "href": "/block/0"
                },
                "manifest": {
                    "href": "/block/0/manifest"
                },
                "self": {
                    "href": "/block/operation/
↪7rSkwgF6BmLmid13jiBJKaaRtgYXS7rtDBFSuNdUNPeo"
                }
            }
        },
        ...
    ],
    "_links": {
        "reverse": {
            "href": "/block/operations?reverse=1"
        },
        "next": {
            "href": "/block/operations?offset=86472,0"
        },
        "self": {
            "href": "/block/operations"
        }
    }
}
```

- 404 (operations not found)

If there aren't any operations, it returns `404`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "operations not found",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/block/operation/{fact_hash}

- It returns the operation information of the operation by *fact hash*.

PATH	METHOD
/block/operation/{fact_hash}	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-operation-value-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-operation-value-v0.0.1",
        "hash": "CtHUdBrLb5cbrkqorSfudS9o4iVMDNafxKiLHZBArHgU",
        "operation": {
            "_hint": "mitum-currency-fee-operation-v0.0.1",
            "hash": "ByDxnBzr116YvesYsFAdn9LR5bw94YWvVEaEFFZukh6H",
            "fact": {
                "_hint": "mitum-currency-fee-operation-fact-v0.0.1",
                "hash": "CtHUdBrLb5cbrkqorSfudS9o4iVMDNafxKiLHZBArHgU",
                "token": "eVQYAAAAAAA=",
                "amounts": [
                    {
                        "_hint": "mitum-currency-amount-v0.0.1",
                        "amount": "1",
                        "currency": "PEN"
                    }
                ]
            }
        },
        "height": 1594489,
```

(continues on next page)

(continued from previous page)

```
        "confirmed_at": "2022-02-02T12:53:44.669Z",
        "reason": null,
        "in_state": true,
        "index": 1
    },
    "_links": {
        "block": {
            "href": "/block/1594489"
        },
        "manifest": {
            "href": "/block/1594489/manifest"
        },
        "operation:{hash}": {
            "templated": true,
            "href": "/block/operation/{hash:(?i)[0-9a-z][0-9a-z]+}"
        },
        "block:{height}": {
            "href": "/block/{height:[0-9]+}",
            "templated": true
        },
        "self": {
            "href": "/block/operation/CtHUdBrLb5cbrkqorSfudS9o4iVMDNafxKiLHZBArHgU"
        }
    }
}
```

- 400 (operation not found)

If the fact hash is wrong, it returns `400`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "title": "invalid hash for operation by hash: invalid; empty hash",
    "detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

2.18.3 Account

/account/{address}

- It returns the latest state of the account by *account address*.

PATH	METHOD
/account/{address}	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-account-value-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-account-value-v0.0.1",
        "hash": "YYWJs2ZEmqvuMHkKco9KwJZL9QUuD9j5QZng5KS4mVR",
        "address": "Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca",
        "keys": {
            "_hint": "mitum-currency-keys-v0.0.1",
            "hash": "Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71y",
            "keys": [
                {
                    "_hint": "mitum-currency-key-v0.0.1",
                    "weight": 50,
                    "key": "kdfdUyAkiG88TVNZ28TV7LoRyLynFzH89btk1ctb9u1Ympu"
                },
                {
                    "_hint": "mitum-currency-key-v0.0.1",
                    "weight": 50,
                    "key": "toPtGPdHCsexeVJcJXykBM14gBEJqc487PmgGVjV3w4vmpu"
                }
            ],
            "threshold": 100
        },
        "balance": [
            {
                "_hint": "mitum-currency-amount-v0.0.1",
                "amount": "10",
                "currency": "CWC"
            }
        ],
        "height": 1198976,
        "previous_height": -2
    },
    "_links": {
        "operations:{offset,reverse}": {
            "templated": true,
            "href": "/account/Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca/operations?
```

(continues on next page)

(continued from previous page)

```
↪offset={offset}&reverse=1"
        },
        "block": {
            "href": "/block/1198976"
        },
        "self": {
            "href": "/account/Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca"
        },
        "operations": {
            "href": "/account/Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca/operations"
        },
        "operations:{offset}": {
            "templated": true,
            "href": "/account/Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca/operations?
↪offset={offset}"
        }
    }
}
```

- 404 (account not found)

It the account address is wrong, it returns `404`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "title": "not found; account, ... not found",
    "detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/account/{address}/operations

- It returns all operations related to the account by *account address*.

PATH	METHOD
/account/{address}/operations	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "",
    "_embedded": [
        {
            "_hint": "mitum-currency-hal-v0.0.1",
            "hint": "mitum-currency-operation-value-v0.0.1",
            "_embedded": {
                "_hint": "mitum-currency-operation-value-v0.0.1",
                "hash": "G57ZwvuAxRA778JGTPz16HSHKhAR6Nb7NegRR2VHNwqd",
                "operation": {
                "fact": {
                    "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                    "hash": "G57ZwvuAxRA778JGTPz16HSHKhAR6Nb7NegRR2VHNwqd",
                    "token": "MjAyMi0wMS0xN1QwNjoxOTo1MS44NTJa",
                    "sender": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
                    "items": [
                        {
                            "_hint": "mitum-currency-create-accounts-multiple-amounts-v0.
↪0.1",
                            "keys": {
                                "_hint": "mitum-currency-keys-v0.0.1",
                                "hash": "CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkf",
                                "keys": [
                                    {
                                        "_hint": "mitum-currency-key-v0.0.1",
                                        "weight": 100,
                                        "key":
↪"kdfdUyAkiG88TVNZ28TV7LoRyLynFzH89btk1ctb9u1Ympu"
                                    }
                                ],
                                "threshold": 100
                            },
                            "amounts": [
                                {
                                    "_hint": "mitum-currency-amount-v0.0.1",
                                    "amount": "1000000000000000000000000000",
                                    "currency": "CWC"
                                },
                                ...
                            ]
                        }
                    ]
                },
                "fact_signs": [
                    {
                        "_hint": "base-fact-sign-v0.0.1",
                        "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
                        "signature":
↪"AN1rKvtUWK3qTmQKr613vW5eQm6qt3fRx4wZwEMdecudX8aP9w73KcbVBxuDPGHWLr9j8nL1MJfdSiMiYXNoM7qpsj59N2S14
↪",
```

(continues on next page)

(continued from previous page)

```
                        "signed_at": "2022-01-17T06:19:51.886Z"
                    }
                ],
                "memo": "",
                "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
                "hash": "6XAxmTGfm8AxK9ey242FU3M1Y6pkzgtK2LoEYTjHrASh"
            },
            "height": 910536,
            "confirmed_at": "2022-01-17T06:19:53.617Z",
            "reason": null,
            "in_state": true,
            "index": 0
        },
        "_links": {
            "block": {
                "href": "/block/910536"
            },
            "manifest": {
                "href": "/block/910536/manifest"
            },
            "new_account:CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkf": {
                "key": "CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkf",
                "address": "CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkfmca",
                "href": "/account/CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkfmca"
                },
                "self": {
                    "href": "/block/operation/
↪G57ZwvuAxRA778JGTPz16HSHKhAR6Nb7NegRR2VHNwqd"
                }
            }
        },
        ...
    ],
    "_links": {
        "next": {
            "href": "/account/CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkfmca/operations?
↪offset=1291226,0"
        },
        "reverse": {
            "href": "/account/CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkfmca/operations?
↪reverse=1"
        },
        "self": {
            "href": "/account/CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkfmca/operations"
        },
        "account": {
            "href": "/account/CCxfWi1oErWX7vbxddAsLx8bXSwR1FUbwEkAJcb8Qmkfmca"
        }
    }
}
```

- 404 (operations not found)

If there are no more operations or there aren't any operations for the account, it returns `404`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "title": "not found; operations not found",
    "detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/accounts?publickey={public_key}

- It returns all accounts which keys contains the *public key* as a key.

PATH	METHOD
/accounts?publickey={public_key}	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "",
    "_embedded": [
        {
            "_hint": "mitum-currency-hal-v0.0.1",
            "hint": "mitum-currency-account-value-v0.0.1",
            "_embedded": {
                "_hint": "mitum-currency-account-value-v0.0.1",
                "hash": "YYWJs2ZEmqvuMHkKco9KwJZL9QUuD9j5QZng5KS4mVR",
                "address": "Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca",
                "keys": {
                    "_hint": "mitum-currency-keys-v0.0.1",
                    "hash": "Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71y",
                    "keys": [
                        {
                            "_hint": "mitum-currency-key-v0.0.1",
                            "weight": 50,
                            "key": "kdfdUyAkiG88TVNZ28TV7LoRyLynFzH89btk1ctb9u1Ympu"
                        },
```

(continues on next page)

(continued from previous page)

```
                        {
                            "_hint": "mitum-currency-key-v0.0.1",
                            "weight": 50,
                            "key": "toPtGPdHCsexeVJcJXykBM14gBEJqc487PmgGVjV3w4vmpu"
                        }
                    ],
                    "threshold": 100
                },
                "height": 1198976,
                "previous_height": -2
            },
            "_links": {
                "operations:{offset,reverse}": {
                    "templated": true,
                    "href": "/account/Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca/
↪operations?offset={offset}&reverse=1"
                },
                "block": {
                    "href": "/block/1198976"
                },
                "self": {
                    "href": "/account/Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca"
                },
                "operations": {
                    "href": "/account/Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca/
↪operations"
                },
                "operations:{offset}": {
                    "href": "/account/Aqv9Gn15zM3j79WgzwVe73RVZ4RbSab7UK9vSpRbF71ymca/
↪operations?offset={offset}",
                    "templated": true
                }
            }
        },
        ...
    ],
    "_links": {
        "next": {
            "href": "/accounts?publickey=kdfdUyAkiG88TVNZ28TV7LoRyLynFzH89btk1ctb9u1Ympu&
↪offset=1279558,JLni2ExjGn87UNUro8G7aeiM97M9LGFo8sQfdvGgxk1mca"
        },
        "self": {
            "href": "/accounts?publickey=kdfdUyAkiG88TVNZ28TV7LoRyLynFzH89btk1ctb9u1Ympu"
        }
    }
}
```

- 400 (accounts not found)

If the public key is invalid(for example, wrong format), it returns `400`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "title": "invalue accounts query: failed to decode publickey, \"...\": failed to␣
↪decode key.BasePublickey: invalid key; pubkey string is empty",
    "detail": ""
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

2.18.4 Builder

/builder/operation

- It returns all available operation types.

PATH	METHOD
/builder/operation	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "",
    "_links": {
        "operation-fact:{create-accounts}": {
            "templated": true,
            "href": "/builder/operation/fact/template/create-accounts"
        },
        "operation-fact:{key-updater}": {
            "templated": true,
            "href": "/builder/operation/fact/template/key-updater"
        },
        "operation-fact:{transfers}": {
            "templated": true,
            "href": "/builder/operation/fact/template/transfers"
        },
        "operation-fact:{currency-register}": {
            "href": "/builder/operation/fact/template/currency-register",
            "templated": true
```

(continues on next page)

(continued from previous page)

```
        },
        "self": {
            "href": "/builder/operation"
        }
    }
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/builder/operation/fact/template/{fact}

- It returns the fact template for the requested operation type.

PATH	METHOD
/builder/operation/fact/template/{fact}	GET

- Available types for `{fact}` can be found by `/builder/operation`.

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
        "hash": "8iBXCwN3q8ZvJJ49iAJEN5ZNAhYAYxV69jDLTB9NyzQW",
        "token": "cmFpc2VkIGJ5",
        "sender": "mothermca",
        "items": [
            {
                "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                "keys": {
                    "_hint": "mitum-currency-keys-v0.0.1",
                    "hash": "DBa8N5of7LZkx8ngH4mVbQmQ2NHDd6gL2mScGfhAEqdd",
                    "keys": [
                        {
                            "_hint": "mitum-currency-key-v0.0.1",
                            "weight": 100,
                            "key": "zzeo6WAS4uqwCss4eRibtLnYHqJM21zhzPbKWQVPttxWmpu"
                        }
                    ],
```

(continues on next page)

(continued from previous page)

```
                    "threshold": 100
                },
                "amounts": [
                    {
                        "_hint": "mitum-currency-amount-v0.0.1",
                        "amount": "-333",
                        "currency": "xXx"
                    }
                ]
            }
        ]
    },
    "_links": {
        "self": {
            "href": "/builder/operation/fact/template/create-accounts"
        }
    },
    "_extra": {
        "default": {
            "token": "cmFpc2VkIGJ5",
            "sender": "mothermca",
            "items.keys.keys.key": "zzeo6WAS4uqwCss4eRibtLnYHqJM21zhzPbKWQVPttxWmpu",
            "items.big": "-333",
            "currency": "xXx"
        }
    }
}
```

- 404 (unknown operation)

If the `{fact}` you requested is wrong, it returns `404`.

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "title": "unknown operation, \"...\"",
    "detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/builder/operation/fact

- It returns the operation message with a fake fact_sign and an operation hash.
- It automatically fills `hash` of fact with a correct fact hash.
- Use a valid fact message as a request json.

PATH	METHOD
/builder/operation/fact	POST

Request Example

- A request json must be a fact message.
- It is available not to fill the `hash` field.

```
{
    "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
    "hash": "",
    "token": "MjAyMS0wOC0yN1QwNjo1MDowNi41OTZa",
    "sender": "ETox5FKJFknprZv7iJk5KnKmqR9kz7fWTEWkHCaDkad3mca",
    "items": [
        {
            "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
            "keys": {
                "_hint": "mitum-currency-keys-v0.0.1",
                "hash": "yAbsevAtgHBT6BXoxJmL2nPveqd1B6kKp2dfAxnoVb1",
                "keys": [
                    {
                        "_hint": "mitum-currency-key-v0.0.1",
                        "key": "bvdEGTsfaG6W3esdY9PjgjrsariGkhU1i3krVWzPaHtYmpu",
                        "weight": 100
                    }
                ],
                "threshold": 100
            },
            "amounts": [
                {
                    "_hint": "mitum-currency-amount-v0.0.1",
                    "amount": "100",
                    "currency": "MCC"
                }
            ]
        }
    ]
}
```

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-create-accounts-operation-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
        "hash": "DJ5eA3wYsE4TZiBM9NrPNVWM8cCuceoZpCUNrSpMNQLa",
        "fact": {
            "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
            "hash": "2SehrkkFaqPDgjD6VyHtiAgBRS5Mc5BMFvK6auALP3Sa",
            "token": "MjAyMS0wOC0yN1QwNjo1MDowNi41OTZa",
            "sender": "ETox5FKJFknprZv7iJk5KnKmqR9kz7fWTEWkHCaDkad3mca",
            "items": [
                {
                    "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                    "keys": {
                        "_hint": "mitum-currency-keys-v0.0.1",
                        "hash": "9dGHYkHV61Nob2UivFHSTrZSYNyjzbZyqvwd2XbQ3w2T",
                        "keys": [
                            {
                                "_hint": "mitum-currency-key-v0.0.1",
                                "weight": 100,
                                "key": "bvdEGTsfaG6W3esdY9PjgjrsariGkhU1i3krVWzPaHtYmpu"
                            }
                        ],
                        "threshold": 100
                    },
                    "amounts": [
                        {
                            "_hint": "mitum-currency-amount-v0.0.1",
                            "amount": "100",
                            "currency": "MCC"
                        }
                    ]
                }
            ]
        },
        "fact_signs": [
            {
                "_hint": "base-fact-sign-v0.0.1",
                "signer": "zzeo6WAS4uqwCss4eRibtLnYHqJM21zhzPbKWQVPttxWmpu",
                "signature": "22UZo26eN",
                "signed_at": "2020-10-08T07:53:26Z"
            }
        ],
        "memo": ""
    },
    "_links": {
        "self": {
            "href": "/builder/operation/fact"
        }
    },
    "_extra": {
        "default": {
```

(continues on next page)

(continued from previous page)

```
            "fact_signs.signer": "zzeo6WAS4uqwCss4eRibtLnYHqJM21zhzPbKWQVPttxWmpu",
            "fact_signs.signature": "22UZo26eN"
        },
        "signature_base": "FW3W9vEA0DQwh6QoRxrjCaSPene+18l1x7v9LUb59tNtaXR1bQ=="
    }
}
```

- 400 (problems in request)

If the fact message you requested is wrong or not available, it returns `400`.

```
{
"_hint": "mitum-currency-problem-v0.0.1",
"type": "https://github.com/spikeekips/mitum-currency/problems/others",
"title": "...",
"detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/builder/operation/sign

- It returns the operation message with a new operation hash.
- It automatically fills `hash` of the operation with the newly generated operation hash.
- So the request operation message is available even though it doesn't have operation hash.

PATH	METHOD
/builder/operation/sign	POST

Request Example

- A request json must be an operation message.
- It is available not to fill the `hash` field. (But fact hash must be correct.)

```
{
    "memo": "",
    "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
    "fact": {
```

(continues on next page)

(continued from previous page)

```
        "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
        "hash": "8yGWvxxQUGUd2tL2EEJSJyDTguXgrDrwwFVAgqnefWp5",
        "token": "MjAyMi0wMi0wM1QwNjoyMTozMi41Njla",
        "sender": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
        "items": [
            {
                "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                "keys": {
                    "_hint": "mitum-currency-keys-v0.0.1",
                    "hash": "GyCVt1JHwrjVmJo3Gjf1wpViDC1sCVjfCY8bEV5aHUrq",
                    "keys": [
                        {
                            "_hint": "mitum-currency-key-v0.0.1",
                            "weight": 100,
                            "key": "hTTVAEnZwaGzs12XLax2M7nY4MAnwykYLA6QpVVEbuuMmpu"
                        }
                    ],
                    "threshold": 100
                },
                "amounts": [
                    {
                        "_hint": "mitum-currency-amount-v0.0.1",
                        "amount": "1000000000000000000000",
                        "currency": "PEN"
                    }
                ]
            }
        ]
    },
    "hash": "",
    "fact_signs": [
        {
            "_hint": "base-fact-sign-v0.0.1",
            "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
            "signature":
↪"AN1rKvtB4BCAHibpYmUZsiPi2abRDJ91Y5qpYpuZuwS2MH1voVSjxCXHhfuTkqAMJCtgEzGtsFaGkjEt9SQucoCia2KDDqQhm
↪",
            "signed_at": "2022-02-03T06:21:32.575Z"
        }
    ]
}
```

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-create-accounts-operation-v0.0.1",
    "_embedded": {
```

(continues on next page)

(continued from previous page)

```
        "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
        "hash": "2UimExSvg5YYywaTqzY69TgAYGFEnKvtU5eHCiptZPLP",
        "fact": {
            "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
            "hash": "8yGWvxxQUGUd2tL2EEJSJyDTguXgrDrwwFVAgqnefWp5",
            "token": "MjAyMi0wMi0wM1QwNjoyMTozMi41Njla",
            "sender": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
            "items": [
                {
                    "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                    "keys": {
                        "_hint": "mitum-currency-keys-v0.0.1",
                        "hash": "GyCVt1JHwrjVmJo3Gjf1wpViDC1sCVjfCY8bEV5aHUrq",
                        "keys": [
                            {
                                "_hint": "mitum-currency-key-v0.0.1",
                                "weight": 100,
                                "key": "hTTVAEnZwaGzs12XLax2M7nY4MAnwykYLA6QpVVEbuuMmpu"
                            }
                        ],
                        "threshold": 100
                    },
                    "amounts": [
                        {
                        "_hint": "mitum-currency-amount-v0.0.1",
                        "amount": "1000000000000000000000",
                        "currency": "PEN"
                        }
                    ]
                }
            ]
        },
        "fact_signs": [
            {
                "_hint": "base-fact-sign-v0.0.1",
                "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
                "signature":
↪"AN1rKvtB4BCAHibpYmUZsiPi2abRDJ91Y5qpYpuZuwS2MH1voVSjxCXHhfuTkqAMJCtgEzGtsFaGkjEt9SQucoCia2KDDqQhm
↪",
                "signed_at": "2022-02-03T06:21:32.575Z"
            }
        ],
        "memo": ""
    },
    "_links": {
        "self": {
            "href": "/builder/operation/sign"
        }
    }
}
```

- 400 (problems in request)

If there is a problem with the request(for example, invalid operation message), it returns `400`.

```
{
"_hint": "mitum-currency-problem-v0.0.1",
"type": "https://github.com/spikeekips/mitum-currency/problems/others",
"title": "...",
"detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/builder/send

- It broadcasts a seal or an operation to the network.
- If broadcast is successful, it returns `200` with the complete seal json.
- However, successful broadcasting doesn't ensure the success of processing the operation.

PATH	METHOD
/builder/send	POST

Request Example

- This API allows to broadcast both operations and seals.
- If the request body is an operation, it makes a new seal containing the operation then broadcasts it.
- If the request body is a seal, it newly signs to the seal and broadcasts it.
- operation

```
{
    "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
    "hash": "2UimExSvg5YYywaTqzY69TgAYGFEnKvtU5eHCiptZPLP",
    "fact": {
        "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
        "hash": "8yGWvxxQUGUd2tL2EEJSJyDTguXgrDrwwFVAgqnefWp5",
        "token": "MjAyMi0wMi0wM1QwNjoyMTozMi41Njla",
        "sender": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
        "items": [
            {
                "_hint": "mitum-currency-create-accounts-single-amount-v0.0.1",
                "keys": {
```

(continues on next page)

(continued from previous page)

```
                    "_hint": "mitum-currency-keys-v0.0.1",
                    "hash": "GyCVt1JHwrjVmJo3Gjf1wpViDC1sCVjfCY8bEV5aHUrq",
                    "keys": [
                        {
                            "_hint": "mitum-currency-key-v0.0.1",
                            "weight": 100,
                            "key": "hTTVAEnZwaGzs12XLax2M7nY4MAnwykYLA6QpVVEbuuMmpu"
                        }
                    ],
                    "threshold": 100
                },
                "amounts": [
                    {
                        "_hint": "mitum-currency-amount-v0.0.1",
                        "amount": "1000000000000000000000",
                        "currency": "PEN"
                    }
                ]
            }
        ]
    },
    "fact_signs": [
    {
        "_hint": "base-fact-sign-v0.0.1",
        "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
        "signature":
↪"AN1rKvtB4BCAHibpYmUZsiPi2abRDJ91Y5qpYpuZuwS2MH1voVSjxCXHhfuTkqAMJCtgEzGtsFaGkjEt9SQucoCia2KDDqQhm
↪",
        "signed_at": "2022-02-03T06:21:32.575Z"
    }
    ],
    "memo": ""
}
```

- seal

```
{
    "_hint": "seal-v0.0.1",
    "hash": "6DrH1RbJHBoKBRFUo33m8foBNti7gSjKg31pgs8L1Cdz",
    "body_hash": "CjjV3HTbTonfkGZWMeXq6rWgBcf8sgRj74i6YdTjNabn",
    "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
    "signature":
↪"AN1rKvszFHPHZVahb17DCx5dzby8c3UBBeV8R2kzPGMiX8e2dceW8n3LifAaPJAHrTs47hF2xiVeyGcqW99j4rwMR1oH4DNeZ
↪",
    "signed_at": "2022-02-03T06:32:28.022166729Z",
    "operations": [
        {
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "GiFDqiwh9j6eqar1yhGGKiT7m8nRaiCW2KqjiAtJQeuu",
            "fact": {
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "J2Kr6rXvZmj2ooTcmvDCba2y2QCZ8dJikSwGpkH5gJBv",
```

(continues on next page)

(continued from previous page)

```
                "token": "MjAyMi0wMi0wM1QwNjozMjoyOC4wMjE4MDg2NzNa",
                "sender": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-multiple-amounts-v0.0.1
↪",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "9Myzqxx5mHxy8oZL1uhvBFQaqwk3Egejh5AaBKsARZka",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"fGZAe2skLHoQ4rhPxbPvjNSjfcPY9292NVyJWX5m4cGYmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "10000",
                                "currency": "PEN"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                "_hint": "base-fact-sign-v0.0.1",
                "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
                "signature":
↪"AN1rKvtYoYJafJUim5BB5sjid8bxNszGB8kuDgbpARnbeGTgUwp2VpVjXS8kbArUVw4axKNb92ZZ4RXmjZn2enHbEAkb6soGL
↪",
                "signed_at": "2022-02-03T06:32:28.022141041Z"
                }
            ],
            "memo": ""
        }
    ]
}
```

Response Example

- 200

```
{
    "_hint": "seal-v0.0.1",
```

(continues on next page)

(continued from previous page)

```
    "hash": "8xNFCxZ6mwgVLXntD7oXapxDLfVXpPDdcjS8Xb4aFQ6m",
    "body_hash": "A1PWmw93mqYd1VXY2ALQFB6qEB7tKQv8AJp1bkAK75QL",
    "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
    "signature":
↪"AN1rKvsxoy63ZDBRqJpz9ps79HHvZMz8jd4yfeTE4v3YFQT5ajoqsZqUF5sTWmACV9R2naBbtXVXamgtw7pPmpSRbkck6NcdF
↪",
    "signed_at": "2022-02-03T06:35:14.742926621Z",
    "operations": [
        {
            "_hint": "mitum-currency-create-accounts-operation-v0.0.1",
            "hash": "GiFDqiwh9j6eqar1yhGGKiT7m8nRaiCW2KqjiAtJQeuu",
            "fact": {
                "_hint": "mitum-currency-create-accounts-operation-fact-v0.0.1",
                "hash": "J2Kr6rXvZmj2ooTcmvDCba2y2QCZ8dJikSwGpkH5gJBv",
                "token": "MjAyMi0wMi0wM1QwNjozMjoyOC4wMjE4MDg2NzNa",
                "sender": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
                "items": [
                    {
                        "_hint": "mitum-currency-create-accounts-multiple-amounts-v0.0.1
↪",
                        "keys": {
                            "_hint": "mitum-currency-keys-v0.0.1",
                            "hash": "9Myzqxx5mHxy8oZL1uhvBFQaqwk3Egejh5AaBKsARZka",
                            "keys": [
                                {
                                    "_hint": "mitum-currency-key-v0.0.1",
                                    "weight": 100,
                                    "key":
↪"fGZAe2skLHoQ4rhPxbPvjNSjfcPY9292NVyJWX5m4cGYmpu"
                                }
                            ],
                            "threshold": 100
                        },
                        "amounts": [
                            {
                                "_hint": "mitum-currency-amount-v0.0.1",
                                "amount": "10000",
                                "currency": "PEN"
                            }
                        ]
                    }
                ]
            },
            "fact_signs": [
                {
                "_hint": "base-fact-sign-v0.0.1",
                "signer": "cnMJqt1Q7LXKqFAWprm6FBC7fRbWQeZhrymTavN11PKJmpu",
                "signature":
↪"AN1rKvtYoYJafJUim5BB5sjid8bxNszGB8kuDgbpARnbeGTgUwp2VpVjXS8kbArUVw4axKNb92ZZ4RXmjZn2enHbEAkb6soGL
↪",
                "signed_at": "2022-02-03T06:32:28.022141041Z"
                }
```

(continues on next page)

(continued from previous page)

```
            ],
            "memo": ""
        }
    ]
}
```

- 400 (problems in request)

If there is a problem with your request(for example, wrong operation or seal), it returns `400`.

```
{
"_hint": "mitum-currency-problem-v0.0.1",
"type": "https://github.com/spikeekips/mitum-currency/problems/others",
"title": "...",
"detail": "..."
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

2.18.5 Currency

/currency

- It returns all currency ids in the network.

PATH	METHOD
/currency	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "",
    "_links": {
        "currency:{currencyid}": {
            "href": "/currency/{currencyid:.*}",
            "templated": true
```

(continues on next page)

(continued from previous page)

```
        },
        "self": {
            "href": "/currency"
        },
        "currency:PEN": {
            "href": "/currency/PEN"
        }
    }
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

/currency/{currency_id}

- It returns the information of the currency by *currency id*.

PATH	METHOD
/currency/{currency_id}	GET

Response Example

- 200

```
{
    "_hint": "mitum-currency-hal-v0.0.1",
    "hint": "mitum-currency-currency-design-v0.0.1",
    "_embedded": {
        "_hint": "mitum-currency-currency-design-v0.0.1",
        "amount": {
            "_hint": "mitum-currency-amount-v0.0.1",
            "amount": "100000000000000000000000000000",
            "currency": "PEN"
        },
        "genesis_account": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
        "policy": {
            "_hint": "mitum-currency-currency-policy-v0.0.1",
            "new_account_min_balance": "10",
            "feeer": {
                "_hint": "mitum-currency-fixed-feeer-v0.0.1",
                "receiver": "8iRVFAPiHKaeznfN3CmNjtFtjYSPMPKLuL6qkaJz8RLumca",
                "amount": "1"
            }
```

(continues on next page)

(continued from previous page)

```
        },
        "aggregate": "1000000000000000000000000000"
    },
    "_links": {
        "currency:{currencyid}": {
            "templated": true,
            "href": "/currency/{currencyid:.*}"
        },
        "block": {
            "href": "/block/0"
        },500
        "operations": {
            "href": "/block/operation/7rSkwgF6BmLmid13jiBJKaaRtgYXS7rtDBFSuNdUNPeo"
        },
        "self": {
            "href": "/currency/PEN"
        }
    }
}
```

- 500

```
{
    "_hint": "mitum-currency-problem-v0.0.1",
    "title": "....",
    "type": "https://github.com/spikeekips/mitum-currency/problems/others",
    "detail": "...."
}
```

2.19 Javascript

This is SDK written in Javascript.

Supported models are,

- Mitum Currency
- Mitum Document

Note that this document introduces how to create operations only for Mitum Currency.

If you would like to check the way to create operations for Mitum Document and the detail explanation for Mitum Currency, please refer to README of mitum-js-util.

2.19.1 Get Started

Prerequisite and Requirements

To use **mitum-js-util** and build it, `npm` or `yarn` should be installed.

Especially, this package has been developed by,

```
$ npm --version
v16.10.0

$ node --version
7.24.0
```

`npm version 16.10.0 or later` is recommended.

Installation

- Using **npm**,

```
$ npm install mitumc
```

- Using **yarn**,

```
$ yarn add mitumc
```

- Using **Git**,

```
$ git clone https://github.com/ProtoconNet/mitum-js-util.git

$ cd mitum-js-util

$ sudo npm install -g

$ cd YOUR_PACKAGE

$ npm link mitumc
```

2.19.2 Make Your First Operation

This tutorial explains how to `create-account` by **mitum-js-util**.

If you want to check how to create `key-updater` and `transfer` operation, go **Support Operations** at the end of this section.

Get Available Account

Before start, you must hold the account registered in the network.

In Mitum Currency, only an existing account can create operations to be stored in a block.

An account consists of the following factors.

```
1. pairs of (public key, weight); aka `keys`
- public key has suffix `mpu`
- The range of each weight should be in 1 <= weight <= 100
- If an account have single public key, the account is called 'single-sig account', or it
↪'s called 'multi-sig account'

2. threshold
- The range of threshold should be in 1 <= threshold <= 100
- The sum of all weights of the account should be over or equal to threshold
```

If you haven't made an account yet, ask other accounts to create your account first.
You can get keypairs for your account in **Details - Get Mitum Keypair** section.
Hand your (public key, weight) pairs and threshold to the account holder who helped create your new account.

For signing, you must remember private keys corresponding each public key of the account. **Don't let not allowed users to know your private key!**
Of course, you must know your account address because you should use the address as `sender`.

You are able to create operations with unauthorized account(like fake keys and address) but those operations will be rejected after broadcasting.

Now, go to the next part to start creating your first operation!

Create Generator

Most of the elements and factors for an operation are created by `Generator`.
For Mitum Currency, use `Generator.mc`.

When declaring a `Generator`, `network id` should be provided.
`network id` is up to each network.

Let's suppose that the network id of the network is `mitum`.

```
import { Generator } from 'mitumc'

const networkId = 'mitum'
const generator = new Generator('mitum')
const currencyGenerator = generator.mc
```

For details about `Generator`, go to **Details - Major Classes** and refer to **Generator**.

In addition, you must have an available account on the network.

Now, you are ready to create operations.

Create Operation Item

Everything to do by an operation is contained in *operation fact*, not in *operation*.
Fact has the basic information such that `sender`, `token`, etc...

Actually, real constructions for the operation are contained in *Item*.
That means you must create items for the operation.

Let's suppose that you want to create an account following conditions below.

```
1. The keys and threshold of the account will be,
    - keys(public key, weight): (kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu, 50),␣
↪(pWoFhRP3C7ocebSRPxTPfeaJZpnyKpEkxQqi6fAD4SHompu, 50)
    - threshold: 100

2. The initial balance of the account will be,
    - balance(currency id, amount): (MCC, 10000), (PEN, 20000)
```

Since the number of keys contained in the account is 2, new account will be a *multi-sig account*.

If every factor of the new account has been decided, create an item!

```
const key1 = currencyGenerator.key("kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu",␣
↪50) // key(pub, weight)
const key2 = currencyGenerator.key("pWoFhRP3C7ocebSRPxTPfeaJZpnyKpEkxQqi6fAD4SHompu",␣
↪50) // key(pub, weight)

const keys = currencyGenerator.keys([key1, key2], 100) // createKeys([key1, key2],␣
↪threshold)
```

(continues on next page)

(continued from previous page)

```
const amount1 = currencyGenerator.amount("MCC", "10000") // amount(currencyId, amount)
const amount2 = currencyGenerator.amount("PEN", "20000") // amount(currencyId, amount)
const amounts = currencyGenerator.amounts([amount1, amount2]); // createAmounts([amount1,
↪ amount2])

const createAccountsItem = currencyGenerator.getCreateAccountsItem(keys, amounts); //␣
↪createCreateAccountsItem(keys, amounts)
```

- First, create each key by `Generator.mc.key(public key, weight)`.
- Second, combine all keys with account threshold by `Generator.mc.keys(key list, threshold)`.
- Third, create each amount by `Generator.mc.amount(currencyId, amount)`.
- Forth, combine all amounts by `Generator.mc.amounts(amount list)`.
- Finally, create an item by `Generator.mc.getCreateAccountsItem(keys, amounts)`

Of course, you can customize the content of items by following constraints.

```
- `Keys` created by `keys` can contain up to 10 key pairs.
- `Amounts` created by `amounts` can contain up to 10 amount pairs.
- Moreover, a `fact` can contain multiple items. The number of items in a fact is up to␣
↪10, either.
```

Create Operation Fact

Fact must have not empty `items`, `sender`, `token`, and `fact hash`.

Don't worry about `token` and `fact hash` because they will be filled automatically by SDK.
The information you must provide is about `items` and `sender`.

The way to create items has been introduced in the section above.

Just be careful that only the account under below conditions can be used as `sender`.

```
1. The account which has been created already.
2. The account which has sufficient balance of currencies in items.
3. The account that you(or owners of the account) know its private keys corresponding␣
↪account public keys.
```

Then, create *fact*!

```
const senderAddress = "CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca" // sender's␣
↪account address; replace with your address
const createAccountsFact = currencyGenerator.getCreateAccountsFact(senderAddress,␣
↪[createAccountsItem]) // getCreateAccountsFact(sender's address, item list)
```

If you want to create fact with multiple items, put them all in item list of `Generator.mc.getCreateAccountsFact(sender's address, item list)` as an array.

Create Operation

Finally, you are in the step to create operation!

Only thing you need to prepare is **sender's private key**. It is used for signing fact.
The signature of a private key is included in `fact_signs` as a **fact signature**.
The sum of weights of all signers in `fact_signs` should exceed or be equal to the `sender`'s threshold.

Only the signatures of the sender account's keys are available to fact_signs!

There is `memo` in operation but it is not necessary. You can enter something if you need, but be careful because that `memo` also affects the `operation hash`.

In this example, suppose that `sender` is a *single-sig account* which means only a single key exists in the sender's account.
If `sender` is a *multi-sig account*, you may add multiple signatures to `fact_signs`.
What key must sign is decided by the account's threshold and keys' weights.

```
const senderPrivateKey = "KxD8T82nfwsUmQu3iMXENm93YTTatGFp1AYDPqTo5e6ycvY1xNXpmpr" //␣
↪sender's private key; replace with your private key

const createAccounts = generator.getOperation(createAccountsFact, "") //␣
↪getOperation(fact, memo)
createAccounts.addSign(senderPrivateKey); // addSign(private key) add fact signature to␣
↪fact_signs
```

Use just `Generator.getOperation(fact, memo)` for create operations, not `Generator.mc.getOperation(fact, memo)`.

Unfortunately, an operation can contain only one fact.

Create Seal

In fact, `operation` itself is enough to create an account.

However, sometimes you may need to wrap multiple operations with a seal.

As mentioned above, one seal can contain multiple operations.

The maximum number of operations in a seal is decided by the policy of nodes.
So check how many operations you can include in a seal before creating seals.

Anyway, it is simple to create a seal with **mitum-js-util**.

What you have to prepare is *private key* from Mitum key package without any conditions.
Any *btc compressed wif* with suffix *mpr* is okay.

```
const anyPrivateKey = "KyK7aMWCbMtCJcneyBZXGG6Dpy2jLRYfx3qp7kxXJjLFnppRYt7wmpr"

const operations = [createAccounts]
const seal = generator.getSeal(anyPrivateKey, operations)
```

Like `getOperation`, use `Generator.getSeal(signer, operation list)`.

Put all operations to wrap in *operation list*.

Support Operations

This section will introduce code example for each operation.

What Mitum Currency operations **mitum-js-util** supports are,

- Create Account
- Key Updater
- Transfer

Create Account

The tutorial for `create-account` have been already explained but it'll be re-introduced in one code-block.

To create a new account you have to prepare,

- The information of the new account: account keys as pairs of (public key, weight), threshold, initial balance as pairs of (currency id, amount).
- Sender's account that has existed already - especially sender's account address and private keys.

As mentioned before, what private keys must sign the fact is up to the threshold and composition of weights.

```
import { Generator } from 'mitumc'

const networkId = 'mitum'
const generator = new Generator('mitum')
const currencyGenerator = generator.mc

const key1 = currencyGenerator.key("kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu", 50)
const key2 = currencyGenerator.key("pWoFhRP3C7ocebSRPxTPfeaJZpnyKpEkxQqi6fAD4SHompu", 50)

const keys = currencyGenerator.keys([key1, key2], 100)

const amount1 = currencyGenerator.amount("MCC", "10000")
const amount2 = currencyGenerator.amount("PEN", "20000")
const amounts = currencyGenerator.amounts([amount1, amount2]);

const createAccountsItem = currencyGenerator.getCreateAccountsItem(keys, amounts);

const senderAddress = "CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca"
const createAccountsFact = currencyGenerator.getCreateAccountsFact(senderAddress,␣
↪[createAccountsItem])

const senderPrivateKey = "KxD8T82nfwsUmQu3iMXENm93YTTatGFp1AYDPqTo5e6ycvY1xNXpmpr"

const createAccounts = generator.getOperation(createAccountsFact, "")
createAccounts.addSign(senderPrivateKey);
```

The detailed explanation was omitted. Refer to the beginning part of 'Make Your First Operation.'.

Key Updater

This operation is to update keys of the account as its name implies.

For example,

```
- I have an single sig account with keys:␣
↪(kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu, 100), threshold: 100
- But I want to replace keys of the account with keys:␣
↪(22ndFZw57ax28ydC3ZxzLJMNX9oMSqAfgauyWhC17pxDpmpu, 50),␣
↪(22wD5RWsRFAr8mHkYmmyUDzKf6VBNgjHcgc3YhKxCvrZDmpu, 50), threshold: 100
- Then you can use key-updater operation to reach the goal!
```

Can I change my account from single-sig to multi-sig? or from multi-sig to single-sig?

Fortunately, of course, you can!

To update keys of the account, you have to prepare,

- The account(target) information you want to change the keys - account address and private keys; what private keys need is up to threshold and key weights.
- New keys: pairs of (public key, weights) and threshold
- Sufficient balance in a currency id to pay a fee.

`create-account` and `transfer` need `item` to create an operation but `key-updater` don't need any item for it. Just create *fact* right now.

```
import { Generator } from 'mitumc'

const networkId = 'mitum'
const generator = new Generator('mitum')
const currencyGenerator = generator.currency

const targetAddress = "JDhSSB3CpRjwM8aF2XX23nTpauv9fLhxTjWsQRm9cJ7umca"
const targetPrivateKey = "KzejtzpPZFdLUXo2hHouamwLoYoPtoffKo5zwoJXsBakKzSvTdbzmpr"

const newPub1 = currencyGenerator.key("22ndFZw57ax28ydC3ZxzLJMNX9oMSqAfgauyWhC17pxDpmpu",
↪ 100)
const newPub2 = currencyGenerator.key("22wD5RWsRFAr8mHkYmmyUDzKf6VBNgjHcgc3YhKxCvrZDmpu",
↪ 100)
const newKeys = currencyGenerator.createKeys([newPub1, newPub2], 100)

const keyUpdaterFact = currencyGenerator.getKeyUpdaterFact(targetAddress, "MCC",␣
```

(continues on next page)

(continued from previous page)

```
↪newKeys) // getKeyUpdaterFact(target address, currency for fee, new keys)

const keyUpdater = generator.getOperation(keyUpdaterFact, "")
keyUpdater.addSign(targetPrivateKey) // only one signature since the account is single-
↪sig
```

- **After updating keys of the account, the keys used before become useless. You should sign operation with private keys of new keypairs of the account.**
- **So record new private keysthreshold somewhere else before sending a key-updater operation to the network.**

Transfer

Finally, you can transfer your tokens to another account.

As other operations, you have to prepare,

- Sender's account information - account address, and private keys
- Pairs of (currency id, amount) to transfer

Like `create-account`, you must create *item* before making *fact*.

Check whether you hold sufficient balance for each currency id to transfer before sending the operation.

Before start, suppose that you want to transfer,

- 1000000 MCC token
- 15000 PEN token

And the receiver is,

- CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca

Note that up to 10 (currency id, amount) pairs can be included in one item.
Moreover, up to 10 items can be included in one fact. However, the receiver for each item should be different.

```
import { Generator } from 'mitumc'

const networkId = 'mitum'
const generator = new Generator('mitum')
```

(continues on next page)

(continued from previous page)

```
const currencyGenerator = generator.currency

const senderPrivateKey = "KzdeJMr8e2fbquuZwr9SEd9e1ZWGmZEj96NuAwHnz7jnfJ7FqHQBmpr"
const senderAddress = "2D5vAb2X3Rs6ZKPjVsK6UHcnGxGfUuXDR1ED1hcvUHqsmca"
const receiverAddress = "CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca"

const amount1 = currencyGenerator.amount("MCC", "1000000")
const amount2 = currencyGenerator.amount("PEN", "15000")
const amounts = currencyGenerator.amounts([amount1, amount2])

const transfersItem = currencyGenerator.getTransfersItem(receiverAddress, amounts) //␣
↪getTransfersItem(receiver address, amounts)
const transfersFact = currencyGenerator.getTransfersFact(senderAddress, [transfersItem])␣
↪// getTransfersFact(sender address, item list)

const transfers = generator.getOperation(transfersFact, "")
transfers.addSign(senderPrivateKey) // suppose sender is single-sig
```

There are other operations that **mitum-js-util** supports, like operations of *Mitum Document*, but this document doesn't provide examples of those operations.
Refer to README if necessary.

2.19.3 Sign

To allow an operation to be stored in blocks, whether signatures of the operation satisfy the **condition** should be checked.

What you have to care about is,

- Has every signature been signed by the private key of the account?
- Is the sum of every weight for each signer greater than or equal to the account threshold?

Of course, there are other conditions each operation must satisfy but we will focus on **signature** (especially about fact signature) in this section.

Let's suppose there is a multi-sig account with 3 keys s.t each weight is 30 and threshold is 50.

That means,

- (pub1, 30)
- (pub2, 30)
- (pub3, 30)
- threshold: 50

When this account wants to send an operation, the operation should include at least two fact signatures of different signers.

1. CASE1: fact signatures signed by pub1's private key and pub2's private key
    1. the sum of pub1's weight and pub2's weight: 60
    2. the sum of weights = 60 > threshold = 50
    3. So the operation with these two fact signatures is available
2. CASE2: fact signatures signed by pub2's private key and pub3's private key
    1. the sum of pub2's weight and pub3's weight: 60
    2. the sum of weights = 60 > threshold = 50
    3. So the operation with these two fact signatures is available
3. CASE3: fact signatures signed by pub1's private key and pub3's private key
    1. the sum of pub1's weight and pub3's weight: 60
    2. the sum of weights = 60 > threshold = 50
    3. So the operation with these two fact signatures is available
4. CASE4: fact signatures signed by pub1's private key, pub2's private key, pub3's private key
    1. the sum of pub1's weight, pub2's weight and pub3's weight: 90
    2. the sum of weights = 90 > threshold = 50
    3. So the operation with these two fact signatures is available

Therefore, you must add multiple signatures to each operation to satisfy the condition. (use `Operation.addSign(private key)`)
Like **CASE4**, it's okay to sign with every private key as long as the sum of their weight >= threshold.

Add Fact Sign to Operation

Besides adding a fact signature when creating the operation, there is another way to add a new fact signature to the operation.

To add a new signature to the operation, you have to prepare,

- Private key to sign - it should be that of the sender of the operation.
- Operation as JS dictionary object, or external JSON file.
- Network ID

First, create `Signer` with `network id` like `Generator`.

```
import { Signer } from 'mitumc'

const networkId = "mitum"
const signKey = "L3CQHoKPJnK61LZhvvvfRouvAjVVabx2RQXHHhPHbBssgcewjgNimpr"
const signer = new Signer(networkId, signKey)
```

Then, sign now!

```
const operationJsonPath = "../createAccount.json" // it's an example; replace with your␣
↪operation path
const operationObject = createAccount.dict() // createAccount is the operation created␣
↪by Generator.createOperation

const signedFromPath = signer.signOperation(operationJsonPath)
const signedFromObject = signer.signOperation(operationObject)
```

`signedFromPath` and `signedFromObject` results in operation with the same fact signatures.

Note that the result operation is not `Operation` object of **mitum-js-util**. It's just a dictionary object.
If you want to add multiple signature at once, you must create another different JSON file then re-sign it with other private keys using `Signer`.

2.19.4 Details

Get Mitum Keypair

We will introduce how to create Mitum keypairs!

Before start, we want to let you know something important; About type suffix.

Address, *private key*, and *public key* in Mitum have specific type suffixes. They are,

- Account Address: `mca`
- Private Key: `mpr`
- Public Key: `mpu`

For example, an single-sig account looks like,

- Account Address: `9XyYKpjad2MSPxR4wfQHvdWrZnk9f5s2zc9Rkdy2KT1gmca`
- Private Key: `L11mKUECzKouwvXwh3eyECsCnvQx5REureuujGBjRuYXbMswFkMxmpr`

- Public Key: `28Hhy6jwkEHx75bNLmG66RQu1LWiZ1vodwRTURtBJhtPWmpu`

There are three methods to create a keypair.

Just Create New Keypair

mitum-js-util will create a random keypair for you!

Use `getNewKeypair()`.

```
import { getNewKeypair } from 'mitumc'

const kp = getNewKeypair() // returns Keypair

kp.getPrivateKey() // KzF4ia7G8in3hm7TzSr5k7cNtx46BdEFTzVdnh82vAopqxJG8rHompr
kp.getPublicKey() // 25jrVNpKr59bYxrWH8eTkbG1iQ8hjvSFKVpfCcDT8oFf8mpu

kp.getRawPrivateKey() // KzF4ia7G8in3hm7TzSr5k7cNtx46BdEFTzVdnh82vAopqxJG8rHo
kp.getRawPublicKey() // 25jrVNpKr59bYxrWH8eTkbG1iQ8hjvSFKVpfCcDT8oFf8mpu
```

Get Keypair From Your Private Key

If you already have your own private key, create keypair with it!

```
import { getKeypairFromPrivateKey } from 'mitumc'

const kp = getKeypairFromPrivateKey(
↪"Kz5b6UMxnRvgL91UvNMuRoTfUEAUw7htW2z4kV2PEZUCVPFmdbXimpr")

kp.getPrivateKey() // Kz5b6UMxnRvgL91UvNMuRoTfUEAUw7htW2z4kV2PEZUCVPFmdbXimpr
kp.getPublicKey() // 239uA6z7MxkZfwp5zYKZ6eBbRWk38AvxeyzfHGQM8o2H8mpu

kp.getRawPrivateKey() // Kz5b6UMxnRvgL91UvNMuRoTfUEAUw7htW2z4kV2PEZUCVPFmdbXi
kp.getRawPublicKey() //239uA6z7MxkZfwp5zYKZ6eBbRWk38AvxeyzfHGQM8o2H8
```

Get Keypair from your seed

You can get a keypair from your seed, too. Even if you don't remember the private key of the keypair, the keypair can be recovered by its seed.
Note that string seed length >= 36.

```
import { getKeypairFromSeed } from 'mitumc'

const kp = getKeypairFromSeed("Thelengthofseedshouldbelongerthan36characters.
↪Thisisaseedfortheexample.")

kp.getPrivateKey() // KynL1wNZjuXvZDboEugU4sWKZ6ck5GTMqtv6eod8Q7C4NaB4kfZPmpr
kp.getPublicKey() // fyLbH5cUwNTihaW2YkJkAzeoLvTNTzf98r8dtCkjXbuqmpu

kp.getRawPrivateKey() // KynL1wNZjuXvZDboEugU4sWKZ6ck5GTMqtv6eod8Q7C4NaB4kfZP
kp.getRawPublicKey() // fyLbH5cUwNTihaW2YkJkAzeoLvTNTzf98r8dtCkjXbuq
```

Get Account Address with Keys

You can calcualte address from threshold, and every (public key, weight) pair of the account.

However, it is not available to get an address if the keys or threshold of the account have changed.
This method is available only for the account that have not changed yet.

The account information for the example is,

- key1: (vmk1iprMrs8V1NkA9DsSL3XQNnUW9SmFL5RCVJC24oFYmpu, 40)
- key2: (29BQ8gcVfJd5hPZCKj335WSe4cyDe7TGrjam7fTrkYNunmpu, 30)
- key3: (uJKiGLBeXF3BdaDMzKSqJ4g7L5kAukJJtW3uuMaP1NLumpu, 30)
- threshold: 100

```
import { Generator } from 'mitumc'

const gn = new Generator('mitum').mc

const key1 = gn.key("vmk1iprMrs8V1NkA9DsSL3XQNnUW9SmFL5RCVJC24oFYmpu", 40)
const key2 = gn.key("29BQ8gcVfJd5hPZCKj335WSe4cyDe7TGrjam7fTrkYNunmpu", 30)
const key3 = gn.key("uJKiGLBeXF3BdaDMzKSqJ4g7L5kAukJJtW3uuMaP1NLumpu", 30)

const keys = gn.keys([key1, key2, key3], 100)

const address = keys.address // this is what you want to get!
```

Major Classes

Generator

`Generator` is the class that helps generate operations for Mitum Currency.

Before you use `Generator`, `network id` must be set.

- For **Mitum Currency**, use `Generator.mc`.
- For **Mitum Document**, use `Generator.md`.

For details of generating operations for **Mitum Document**, refer to README.

```
import { Generator } from 'mitumc'

const networkId = 'mitum'
const generator = new Generator(networkId)

const currencyGenerator = generator.mc
const documentGenerator = generator.md
```

All methods of `Generator` provides are,

```
/* For Mitum Currency */
Generator.mc.key(key, weight) // 1 <= $weight <= 100
Generator.mc.amount(currencyId, amount) // typeof $amount === "string"
Generator.mc.keys(keys, threshold) // 1 <= $threshold <= 100
Generator.mc.amounts(amounts)
Generator.mc.getCreateAccountsItem(keys, amounts)
Generator.mc.getTransfersItem(receiver, amounts)
Generator.mc.getCreateAccountsFact(sender, items)
Generator.mc.getKeyUpdaterFact(target, currencyId, keys)
Generator.mc.getTransfersFact(sender, items)

/* For Mitum Document */
Generator.md.getCreateDocumentsItem(document, currencyId)
Generator.md.getUpdateDocumentsItem(document, currencyId)
Generator.md.getCreateDocumentsFact(sender, items)
Generator.md.getUpdateDocumentsFact(sender, items)

/* For Blocksign*/
Generator.md.bs.user(address, signcode, signed)
Generator.md.bs.document(documentId, owner, fileHash, creator, title, size, signers)
Generator.md.bs.getSignDocumentsItem(documentId, owner, currencyId)
Generator.md.bs.getSignDocumentsFact(sender, items)

/* For Blockcity */
Generator.md.bc.candidate(address, nickname, manifest, count)
Generator.md.bc.userStatistics(hp, strength, agility, dexterity, charisma intelligence,␣
↪vital)
Generator.md.bc.userDocument(documentId, owner, gold, bankGold, userStatistics)
Generator.md.bc.landDocument(documentId, owner, address, area, renter, account, rentDate,
↪ period)
Generator.md.bc.voteDocument(documentId, owner, round, endTime, candidates, bossName,␣
↪account, office)
Generator.md.bc.historyDocument(documentId, owner, name, account, date, usage,␣
```

(continues on next page)

(continued from previous page)

```
↪application)

/* Common */
Generator.getOperation(fact, memo)
Generator.getSeal(signKey, operations)
```

Signer

Signer is the class for adding new fact signature to already create operations.

Like Generator, network id must be set.

You have to prepare *private key* to sign, too.

Signer provides only one method, that is,

```
Signer.signOperation(operation)
```

To check the exact usage of Signer, go back to **Make Your First Operation - Sign**.

JSONParser

This class is constructed just for convenience.
If you would like to use other js packages to export Operation to file or to print it in JSON format, you don't need to use JSONParser of **mitum-js-util**.

```
import { Generator, JSONParser } from 'mitumc'

const generator = new Generator('mitum')
const currencyGenerator = generator.mc

// ... omitted
// ... create operations
// ... refer to above `Make Your First Operation`
// ... suppose you have already made operations - createAccount, keyUpdater, transfer␣
↪and a seal - seal

JSONParser.toJSONString(createAccount.dict()) // print operation createAccount in JSON
JSONParser.toJSONString(keyUpdater.dict()) // print operation keyUpdater in JSON
JSONParser.toJSONString(transfer.dict()) // print operation transfer in JSON
JSONParser.toJSONString(seal) // print seal seal in JSON
```

(continues on next page)

(continued from previous page)

```
JSONParser.getFile(createAccount.dict(), 'createAccount.json'); // getFile(dict object,␣
↪file path)
JSONParser.getFile(keyUpdater.dict(), 'keyUpdater.json');
JSONParser.getFile(transfer.dict(), 'transfer.json');
JSONParser.getFile(seal, 'seal.json');
```

2.20 Python

This is SDK written in Python.

Supported models are,

- Mitum Currency
- Mitum Document

Note that this document introduces how to create operations only for Mitum Currency.

If you would like to check the way to create operations for Mitum Document and the detail explanation for Mitum Currency, please refer to README of mitum-py-util.

2.20.1 Get Started

Prerequisite and Requirements

This package has been developed by,

```
$ python --version
Python 3.9.2
```

`python 3.9 or later` is recommended.

Installation

- Using **Git**,

```
$ git clone https://github.com/ProtoconNet/mitum-py-util.git

$ cd mitum-py-util

$ python setup.py install
```

If setup.py doesn't work properly, please just install necessary packages with requirements.txt before running setup.py.

```
$ pip install -r requirements.txt
```

2.20.2 Make Your First Operation

This tutorial explains how to `create-account` by **mitum-py-util**.

If you want to check how to create `key-updater` and `transfer` operation, go **Support Operations** at the end of this section.

Get Available Account

Before start, you must hold the account registered in the network.

In Mitum Currency, only an existing account can create operations to be stored in a block.

An account consists of the following factors.

```
1. pairs of (public key, weight); aka `keys`
- public key has suffix `mpu`
- The range of each weight should be in 1 <= weight <= 100
- If an account have single public key, the account is called 'single-sig account', or it
↪'s called 'multi-sig account'

2. threshold
- The range of threshold should be in 1 <= threshold <= 100
- The sum of all weights of the account should be over or equal to threshold
```

If you haven't made an account yet, ask other accounts to create your account first.
You can get keypairs for your account in **Details - Get Mitum Keypair** section.
Hand your (public key, weight) pairs and threshold to the account holder who helped create your new account.

For signing, you must remember private keys corresponding each public key of the account. **Don't let not allowed users to know your private key!**
Of course, you must know your account address because you should use the address as `sender`.

You are able to create operations with unauthorized account(like fake keys and address) but those operations will be rejected after broadcasting.

Now, go to the next part to start creating your first operation!

Create Generator

Most of the elements and factors for an operation are created by `Generator`.
For Mitum Currency, use `Generator.mc`.

When declaring a `Generator`, `network id` should be provided.
`network id` is up to each network.

Let's suppose that the network id of the network is `mitum`.

```
from mitumc import Generator

networkId = 'mitum'
generator = Generator('mitum')
currencyGenerator = generator.mc
```

For details about `Generator`, go to **Details - Major Classes** and refer to **Generator**.

In addition, you must have an available account on the network.

Now, you are ready to create operations.

Create Operation Item

Everything to do by an operation is contained in *operation fact*, not in *operation*.
Fact has the basic information such that `sender`, `token`, etc...

Actually, real constructions for the operation are contained in *Item*.
That means you must create items for the operation.

Let's suppose that you want to create an account following conditions below.

```
1. The keys and threshold of the account will be,
    - keys(public key, weight): (kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu, 50),␣
↪(pWoFhRP3C7ocebSRPxTPfeaJZpnyKpEkxQqi6fAD4SHompu, 50)
    - threshold: 100

2. The initial balance of the account will be,
    - balance(currency id, amount): (MCC, 10000), (PEN, 20000)
```

Since the number of keys contained in the account is 2, new account will be a *multi-sig account*.

If every factor of the new account has been decided, create an item!

```
key1 = currencyGenerator.key("kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu", 50) #␣
↪key(public key, weight)
key2 = currencyGenerator.key("pWoFhRP3C7ocebSRPxTPfeaJZpnyKpEkxQqi6fAD4SHompu", 50)
keys = currencyGenerator.keys([key1, key2], 100) # keys(keyList, threshold)

amount1 = currencyGenerator.amount(10000, 'MCC') # amount(amount, currency id)
amount1 = currencyGenerator.amount(20000, 'PEN')
amounts = currencyGenerator.amounts([amount]) # amounts(amountList)

createAccountsItem = currencyGenerator.getCreateAccountsItem(keys, amounts)
```

- First, create each key by `Generator.mc.key(public key, weight)`.
- Second, combine all keys with account threshold by `Generator.mc.keys(key list, threshold)`.
- Third, create each amount by `Generator.mc.amount(amount, currencyId)`.
- Forth, combine all amounts by `Generator.mc.amounts(amount list)`.
- Finally, create an item by `Generator.mc.getCreateAccountsItem(keys, amounts)`

Of course, you can customize the content of items by following constraints.

```
- `Keys` created by `keys` can contain up to 10 key pairs.
- `Amounts` created by `amounts` can contain up to 10 amount pairs.
- Moreover, a `fact` can contain multiple items. The number of items in a fact is up to␣
↪10, either.
```

Create Operation Fact

Fact must have not empty `items`, `sender`, `token`, and `fact hash`.

Don't worry about `token` and `fact hash` because they will be filled automatically by SDK.
The information you must provide is about `items` and `sender`.

The way to create items has been introduced in the section above.

Just be careful that only the account under below conditions can be used as `sender`.

```
1. The account which has been created already.
2. The account which has sufficient balance of currencies in items.
3. The account that you(or owners of the account) know its private keys corresponding␣
↪account public keys.
```

Then, create *fact*!

```
senderAddress = "CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca" # sender's account␣
↪address; replace with your address
createAccountsFact = currencyGenerator.getCreateAccountsFact(senderAddress,␣
↪[createAccountsItem]) # createCreateAccountsFact(sender's address, item list)
```

If you want to create fact with multiple items, put them all in item list of `Generator.mc.getCreateAccountsFact(sender's address, item list)` as an array.

Create Operation

Finally, you are in the step to create operation!

Only thing you need to prepare is **sender's private key**. It is used for signing fact.
The signature of a private key is included in `fact_signs` as a **fact signature**.
The sum of weights of all signers in `fact_signs` should exceed or be equal to the `sender`'s threshold.

Only the signatures of the sender account's keys are available to fact_signs!

There is `memo` in operation but it is not necessary. You can enter something if you need, but be careful because that `memo` also affects the `operation hash`.

In this example, suppose that `sender` is a *single-sig account* which means only a single key exists in the sender's account.
If `sender` is a *multi-sig account*, you may add multiple signatures to `fact_signs`.
What key must sign is decided by the account's threshold and keys' weights.

```
senderPrivateKey = "KxD8T82nfwsUmQu3iMXENm93YTTatGFp1AYDPqTo5e6ycvY1xNXpmpr" # sender's␣
↪private key; replace with your private key

createAccounts = generator.getOperation(createAccountsFact, "") # getOperation(fact,␣
↪memo)
createAccounts.addFactSign(senderPrivateKey); # addFactSign(private key) add fact␣
↪signature to fact_signs
```

Use just `Generator.createOperation(fact, memo)` for create operations, not `Generator.currency.createOperation(fact, memo)`.

Unfortunately, an operation can contain only one fact.

Create Seal

In fact, `operation` itself is enough to create an account.

However, sometimes you may need to wrap multiple operations with a seal.

As mentioned above, one seal can contain multiple operations.

The maximum number of operations in a seal is decided by the policy of nodes.
So check how many operations you can include in a seal before creating seals.

Anyway, it is simple to create a seal with **mitum-py-util**.

What you have to prepare is *private key* from Mitum key package without any conditions.
Any *btc compressed wif* with suffix *mpr* is okay.

```
signKey = "L1V19fBjhnxNyfuXLWw6Y5mjFSixzdsZP4obkXEERskGQNwSgdm1mpr"

operations = [createAccounts]
seal = generator.getSeal(signKey, operations)
```

Like `getOperation`, use `Generator.getSeal(signer, operation list)`.

Put all operations to wrap in *operation list*.

Support Operations

This section will introduce code example for each operation.

What Mitum Currency operations **mitum-py-util** supports are,

- Create Account
- Key Updater
- Transfer

Create Account

The tutorial for `create-account` have been already explained but it'll be re-introduced in one code-block.

To create a new account you have to prepare,

- The information of the new account: account keys as pairs of (public key, weight), threshold, initial balance as pairs of (currency id, amount).
- Sender's account that has existed already - especially sender's account address and private keys.

As mentioned before, what private keys must sign the fact is up to the threshold and composition of weights.

```
from mitumc import Generator

senderPrivateKey = "L1V19fBjhnxNyfuXLWw6Y5mjFSixzdsZP4obkXEERskGQNwSgdm1mpr"
senderAddress = "5fbQg8K856KfvzPiGhzmBMb6WaL5AsugUnfutgmWECPbmca"

generator = Generator('mitum')
gn = generator.mc

key = gn.key("2177RF13ZZXpdE1wf7wu5f9CHKaA2zSyLW5dk18ExyJ84mpu", 100)
keys = gn.keys([key], 100)

amount = gn.amount(100, 'MCC')
amounts = gn.amounts([amount])

createAccountsItem = gn.getCreateAccountsItem(keys, amounts)
createAccountsFact = gn.getCreateAccountsFact(srcAddr, [createAccountsItem])

createAccounts = generator.getOperation(createAccountsFact, "")
createAccounts.addFactSign(srcPriv)
```

The detailed explanation was omitted. Refer to the beginning part of 'Make Your First Operation.'.

Key Updater

This operation is to update keys of the account as its name implies.

For example,

```
- I have an single sig account with keys:␣
↪(kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu, 100), threshold: 100
- But I want to replace keys of the account with keys:␣
```

(continues on next page)

(continued from previous page)

```
↪(22ndFZw57ax28ydC3ZxzLJMNX9oMSqAfgauyWhC17pxDpmpu, 50),␣
↪(22wD5RWsRFAr8mHkYmmyUDzKf6VBNgjHcgc3YhKxCvrZDmpu, 50), threshold: 100
- Then you can use key-updater operation to reach the goal!
```

Can I change my account from single-sig to multi-sig? or from multi-sig to single-sig?

Fortunately, of course, you can!

To update keys of the account, you have to prepare,

- The account(target) information you want to change the keys - account address and private keys; what private keys need is up to threshold and key weights.
- New keys: pairs of (public key, weights) and threshold
- Sufficient balance in a currency id to pay a fee.

`create-account` and `transfer` need `item` to create an operation but `key-updater` don't need any item for it. Just create *fact* right now.

```
from mitumc import Generator

targetPrivateKey = "KzejtzpPZFdLUXo2hHouamwLoYoPtoffKo5zwoJXsBakKzSvTdbzmpr"
targetAddress = "JDhSSB3CpRjwM8aF2XX23nTpauv9fLhxTjWsQRm9cJ7umca"

generator = Generator('mitum')
gn = generator.mc

key1 = gn.key("22ndFZw57ax28ydC3ZxzLJMNX9oMSqAfgauyWhC17pxDpmpu", 50)
key2 = gn.key("22wD5RWsRFAr8mHkYmmyUDzKf6VBNgjHcgc3YhKxCvrZDmpu", 50)
keys = gn.keys([key1, key2], 100)

keyUpdaterFact = gn.getKeyUpdaterFact(targetAddress, keys, "MCC") #␣
↪getKeyUpdaterFact(target address, new keys, currency id for fee)

keyUpdater = generator.getOperation(keyUpdaterFact, "")
keyUpdater.addFactSign(targetPrivateKey)
```

- **After updating keys of the account, the keys used before become useless. You should sign operation with private keys of new keypairs of the account.**
- **So record new private keysthreshold somewhere else before sending a key-updater operation to the network.**

Transfer

Finally, you can transfer your tokens to another account.

As other operations, you have to prepare,

- Sender's account information - account address, and private keys
- Pairs of (currency id, amount) to transfer

Like `create-account`, you must create *item* before making *fact*.

Check whether you hold sufficient balance for each currency id to transfer before sending the operation.

Before start, suppose that you want to transfer,

- 1000000 MCC token
- 15000 PEN token

And the receiver is,

- CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca

Note that up to 10 (currency id, amount) pairs can be included in one item.
Moreover, up to 10 items can be included in one fact. However, the receiver for each item should be different.

```
from mitumc import Generator

generator = Generator('mitum')
gn = generator.mc

senderPrivateKey = "KzdeJMr8e2fbquuZwr9SEd9e1ZWGmZEj96NuAwHnz7jnfJ7FqHQBmpr"
senderAddress = "2D5vAb2X3Rs6ZKPjVsK6UHcnGxGfUuXDR1ED1hcvUHqsmca"
receiverAddress = "CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca"

amount = gn.amount(1000000, 'MCC')
amount = gn.amount(15000, 'PEN')
amounts = gn.amounts([amount1, amount2])

transfersItem = gn.getTransfersItem(receiverAddress, amounts) #␣
↪getTransfersItem(receiver address, amounts)
transfersFact = gn.getTransfersFact(senderAddress, [transfersItem]) #␣
↪getTransfersFact(sender addrewss, item list)
```

(continues on next page)

(continued from previous page)

```
transfers = generator.getOperation(transfersFact, "")
transfers.addFactSign(senderPrivateKey)
```

There are other operations that **mitum-py-util** supports, like operations of *Mitum Document*, but this document doesn't provide examples of those operations.
Refer to README if necessary.

2.20.3 Sign

To allow an operation to be stored in blocks, whether signatures of the operation satisfy the **condition** should be checked.

What you have to care about is,

- Has every signature been signed by the private key of the account?
- Is the sum of every weight for each signer greater than or equal to the account threshold?

Of course, there are other conditions each operation must satisfy but we will focus on **signature** (especially about fact signature) in this section.

Let's suppose there is a multi-sig account with 3 keys s.t each weight is 30 and threshold is 50.

That means,

- (pub1, 30)
- (pub2, 30)
- (pub3, 30)
- threshold: 50

When this account wants to send an operation, the operation should include at least two fact signatures of different signers.

1. CASE1: fact signatures signed by pub1's private key and pub2's private key
   1. the sum of pub1's weight and pub2's weight: 60
   2. the sum of weights = 60 > threshold = 50
   3. So the operation with these two fact signatures is available
2. CASE2: fact signatures signed by pub2's private key and pub3's private key
   1. the sum of pub2's weight and pub3's weight: 60

   2. the sum of weights = 60 > threshold = 50
   3. So the operation with these two fact signatures is available
3. CASE3: fact signatures signed by pub1's private key and pub3's private key
   1. the sum of pub1's weight and pub3's weight: 60
   2. the sum of weights = 60 > threshold = 50
   3. So the operation with these two fact signatures is available
4. CASE4: fact signatures signed by pub1's private key, pub2's private key, pub3's private key
   1. the sum of pub1's weight, pub2's weight and pub3's weight: 90
   2. the sum of weights = 90 > threshold = 50
   3. So the operation with these two fact signatures is available

Therefore, you must add multiple signatures to each operation to satisfy the condition. (use `Operation.addFactSign(private key)`)
Like **CASE4**, it's okay to sign with every private key as long as the sum of their weight >= threshold.

Add Fact Sign to Operation

Besides adding a fact signature when creating the operation, there is another way to add a new fact signature to the operation.

To add a new signature to the operation, you have to prepare,

- Private key to sign - it should be that of the sender of the operation.
- Operation as JS dictionary object, or external JSON file.
- Network ID

First, create `Signer` with `network id` like `Generator`.

```
from mitumc import Signer

networkId = 'mitum'
signKey = 'L1V19fBjhnxNyfuXLWw6Y5mjFSixzdsZP4obkXEERskGQNwSgdm1mpr'
signer = Signer(networkId, signKey)
```

Then, sign now!

```
signed = signer.signOperation('operation.json') # signOperation(filePath)
```

Note that the result operation is not `Operation` object of **mitum-py-util**. It's just a dictionary object.
If you want to add multiple signature at once, you must create another different JSON file then re-sign it with other private keys using `Signer`.

2.20.4 Details

Get Mitum Keypair

We will introduce how to create Mitum keypairs!

Before start, we want to let you know something important; About type suffix.

Address, *private key*, and *public key* in Mitum have specific type suffixes. They are,

- Account Address: `mca`
- Private Key: `mpr`
- Public Key: `mpu`

For example, an single-sig account looks like,

- Account Address: `9XyYKpjad2MSPxR4wfQHvdWrZnk9f5s2zc9Rkdy2KT1gmca`
- Private Key: `L11mKUECzKouwvXwh3eyECsCnvQx5REureuujGBjRuYXbMswFkMxmpr`
- Public Key: `28Hhy6jwkEHx75bNLmG66RQu1LWiZ1vodwRTURtBJhtPWmpu`

There are three methods to create a keypair.

Just Create New Keypair

mitum-py-util will create a random keypair for you!

Use `getNewKeypair()`.

```
from mitumc.key import getNewKeypair

# get new Keypair
kp = getNewKeypair() # returns BTCKeyPair
kp.privateKey # KzafpyGojcN44yme25UMGvZvKWdMuFv1SwEhsZn8iF8szUz16jskmpr
kp.publicKey # 24TbbrNYVngpPEdq6Zc5rD1PQSTGQpqwabB9nVmmonXjqmpu
```

Get Keypair From Your Private Key

If you already have your own private key, create keypair with it!

```python
from mitumc.key import getKeypairFromPrivateKey

# get Keypair from your private key
pkp = getKeypairFromPrivateKey("L2ddEkdgYVBkhtdN8HVXLZk5eAcdqXxecd17FDTobVeFfZNPk2ZDmpr")
```

Get Keypair From Your Seed

You can get a keypair from your seed, too. Even if you don't remember the private key of the keypair, the keypair can be recovered by its seed.
Note that string seed length >= 36.

```python
from mitumc.key import getKeypairFromSeed

# get Keypair from your seed
skp = getKeypairFromSeed("Thisisaseedforthisexample.len(seed)>=36.")
```

Get Account Address with Keys

You can calcualte address from threshold, and every (public key, weight) pair of the account.

However, it is not available to get an address if the keys or threshold of the account have changed.
This method is available only for the account that have not changed yet.

The account information for the example is,

- key1: (vmk1iprMrs8V1NkA9DsSL3XQNnUW9SmFL5RCVJC24oFYmpu, 40)
- key2: (29BQ8gcVfJd5hPZCKj335WSe4cyDe7TGrjam7fTrkYNunmpu, 30)
- key3: (uJKiGLBeXF3BdaDMzKSqJ4g7L5kAukJJtW3uuMaP1NLumpu, 30)
- threshold: 100

```python
from mitumc import Generator

gn = Generator('mitum').mc

pub1 = "vmk1iprMrs8V1NkA9DsSL3XQNnUW9SmFL5RCVJC24oFYmpu"
pub2 = "29BQ8gcVfJd5hPZCKj335WSe4cyDe7TGrjam7fTrkYNunmpu"
pub3 = "uJKiGLBeXF3BdaDMzKSqJ4g7L5kAukJJtW3uuMaP1NLumpu"

key1 = gn.key(pub1, 40)
```

(continues on next page)

(continued from previous page)

```
key2 = gn.key(pub2, 30)
key3 = gn.key(pub3, 30)

keys = gn.keys([key1, key2, key3], 100)
address = keys.address # your address
```

Major Classes

Generator

`Generator` is the class that helps generate operations for Mitum Currency.

Before you use `Generator`, `network id` must be set.

- For **Mitum Currency**, use `Generator.mc`.
- For **Mitum Document**, use `Generator.md`.

For details of generating operations for **Mitum Document**. refer to README.

```
from mitumc import Generator

generator = Generator('mitum')
currencyGenerator = generator.mc
documentGenerator = generator.md
```

All methods of `Generator` provides are,

```
# For Mitum Currency
Generator.mc.key(key, weight) # 1 <= $weight <= 100
Generator.mc.amount(currencyId, amount)
Generator.mc.keys(keys, threshold) # 1 <= $threshold <= 100
Generator.mc.amounts(amounts)
Generator.mc.getCreateAccountsItem(keys, amounts)
Generator.mc.getTransfersItem(receiver, amounts)
Generator.mc.getCreateAccountsFact(sender, items)
Generator.mc.getKeyUpdaterFact(target, currencyId, keys)
Generator.mc.getTransfersFact(sender, items)

# For Mitum Document
Generator.md.getCreateDocumentsItem(document, currencyId)
Generator.md.getUpdateDocumentsItem(document, currencyId)
Generator.md.getCreateDocumentsFact(sender, items)
Generator.md.getUpdateDocumentsFact(sender, items)
```

(continues on next page)

(continued from previous page)

```
# For Blocksign
Generator.md.bs.user(address, signcode, signed)
Generator.md.bs.document(documentId, owner, fileHash, creator, title, size, signers)
Generator.md.bs.getSignDocumentsItem(documentId, owner, currencyId)
Generator.md.bs.getSignDocumentsFact(sender, items)

# For Blockcity
Generator.md.bc.candidate(address, nickname, manifest, count)
Generator.md.bc.userStatistics(hp, strength, agility, dexterity, charisma intelligence,␣
↪vital)
Generator.md.bc.userDocument(documentId, owner, gold, bankGold, userStatistics)
Generator.md.bc.landDocument(documentId, owner, address, area, renter, account, rentDate,
↪ period)
Generator.md.bc.voteDocument(documentId, owner, round, endTime, candidates, bossName,␣
↪account, office)
Generator.md.bc.historyDocument(documentId, owner, name, account, date, usage,␣
↪application)

# Common
Generator.getOperation(fact, memo)
Generator.getSeal(signKey, operations)
```

Signer

`Signer` is the class for adding new fact signature to already create operations.

Like `Generator`, `network id` must be set.

You have to prepare *private key* to sign, too.

`Signer` provides only one method, that is,

```
Signer.signOperation(operation)
```

To check the exact usage of `Signer`, go back to **Make Your First Operation - Sign**.

JSONParser

This class is constructed just for convenience.
If you would like to use other python packages to export `Operation` to file or to print it in JSON format, you don't need to use `JSONParser` of **mitum-py-util**.

```python
from mitumc import JSONParser

# ... omitted
# ... create operations
# ... refer to above `Make Your First Operation`
# ... suppose you have already made operations - createAccount, keyUpdater, transfer and␣
↪a seal - seal

JSONParser.toString(createAccount.dict()) # print operation createAccount in JSON
JSONParser.toString(keyUpdater.dict()) # print operation keyUpdater in JSON
JSONParser.toString(transfer.dict()) # print operation transfer in JSON
JSONParser.toString(seal) # print seal seal in JSON

JSONParser.toFile(createAccount.dict(), 'createAccount.json') # toFile(dict object, file␣
↪path)
JSONParser.toFile(keyUpdater.dict(), 'keyUpdater.json')
JSONParser.toFile(transfer.dict(), 'transfer.json')
JSONParser.toFile(seal, 'seal.json')
```

2.21 Java

This is SDK written in Java.

Supported models are,

- Mitum Currency
- Mitum Document - blocksign, blockcity

Note that this document introduces how to create operations only for Mitum Currency.

If you would like to check the way to create operations for Mitum Document and the detail explanation for Mitum Currency, please refer to README of mitum-java-util.

2.21.1 Get Started

Prerequisite and Requirements

This package has been developed by,

```
$ java -version
java 17.0.1 2021-10-19 LTS
Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-LTS-39, mixed mode, sharing)

$ javac -version
javac 17.0.1
```

And this package is using this external Java Library.

- bitcoinj v0.14.7

Installation

Download jar file from the repository.

Now, the latest version is `mitum-java-util-3.0.0-jdk17.jar`.

Using *Gradle*,

```
implementation files('./lib/mitum-java-util-3.0.0-jdk17.jar')
```

2.21.2 Make Your First Operation

This tutorial explains how to `create-account` by **mitum-java-util**.

If you want to check how to create `key-updater` and `transfer` operation, go **Support Operations** at the end of this section.

Get Available Account

Before start, you must hold the account registered in the network.

In Mitum Currency, only an existing account can create operations to be stored in a block.

An account consists of the following factors.

```
1. pairs of (public key, weight); aka `keys`
- public key has suffix `mpu`
- The range of each weight should be in 1 <= weight <= 100
- If an account have single public key, the account is called 'single-sig account', or it
↪'s called 'multi-sig account'

2. threshold
- The range of threshold should be in 1 <= threshold <= 100
- The sum of all weights of the account should be over or equal to threshold
```

If you haven't made an account yet, ask other accounts to create your account first.
You can get keypairs for your account in **Details - Get Mitum Keypair** section.
Hand your (public key, weight) pairs and threshold to the account holder who helped create your new account.

For signing, you must remember private keys corresponding each public key of the account. **Don't let not allowed users to know your private key!**
Of course, you must know your account address because you should use the address as `sender`.

You are able to create operations with unauthorized account(like fake keys and address) but those operations will be rejected after broadcasting.

Now, go to the next part to start creating your first operation!

Create Generator

Most of the elements and factors for an operation are created by `Generator`.
For Mitum Currency, use `Generator.mc()`.

When declaring a `Generator`, `network id` should be provided.
`network id` is up to each network.

Let's suppose that the network id of the network is `mitum`.

```
/*
import org.mitumc.sdk.Generator
import org.mitumc.sdk.operation.currency.CurrencyGenerator;
*/
String id = "mitum";
Generator generator = Generator.get(id);
CurrencyGenerator cgn = generator.mc();
```

For details about `Generator`, go to **Details - Major Classes** and refer to **Generator**.

In addition, you must have an available account on the network.

Now, you are ready to create operations.

Create Operation Item

Everything to do by an operation is contained in *operation fact*, not in *operation*.
Fact has the basic information such that `sender`, `token`, etc…

Actually, real constructions for the operation are contained in *Item*.
That means you must create items for the operation.

Let's suppose that you want to create an account following conditions below.

```
1. The keys and threshold of the account will be,
    - keys(public key, weight): (kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu, 50),␣
↪(pWoFhRP3C7ocebSRPxTPfeaJZpnyKpEkxQqi6fAD4SHompu, 50)
    - threshold: 100

2. The initial balance of the account will be,
    - balance(currency id, amount): (MCC, 10000), (PEN, 20000)
```

Since the number of keys contained in the account is 2, new account will be a *multi-sig account*.

If every factor of the new account has been decided, create an item!

```
/*
import org.mitumc.sdk.key.*;
import org.mitumc.sdk.operation.currency.*;
*/
Key key1 = generator.mc().key("kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu", 50); //␣
```

(continues on next page)

(continued from previous page)

```
↪newKey(public key, weight)
Key key2 = generator.mc().key("pWoFhRP3C7ocebSRPxTPfeaJZpnyKpEkxQqi6fAD4SHompu", 50);
Keys keys = generator.mc().keys(new Key[]{ key1, key2 }, 100); // newKeys(key list,␣
↪threshold)

Amount amount1 = generator.mc().amount("MCC", "10000"); // newAmount(currency id, amount)
Amount amount2 = generator.mc().amount("PEN", "20000");

CreateAccountsItem item = generator.mc().getCreateAccountsItem(keys, new Amount[]{␣
↪amount1, amount2 }); // newCreateAccountsItem(keys, amount list)
```

- First, create each key by `Generator.mc().key(public key, weight)`.
- Second, combine all keys with account threshold by `Generator.mc().keys(key list, threshold)`.
- Third, create each amount by `Generator.mc().amount(currencyId, amount)`.
- Finally, create an item by `Generator.mc().getCreateAccountsItem(keys, amount list)`

Of course, you can customize the content of items by following constraints.

```
- `Keys` created by `keys` can contain up to 10 key pairs.
- `Amount list` s.t each amount created by `amounts` can contain up to 10 in one item.
- Moreover, a `fact` can contain multiple items. The number of items in a fact is up to␣
↪10, either.
```

Create Operation Fact

Fact must have not empty `items`, `sender`, `token`, and `fact hash`.

Don't worry about `token` and `fact hash` because they will be filled automatically by SDK.
The information you must provide is about `items` and `sender`.

The way to create items has been introduced in the section above.

Just be careful that only the account under below conditions can be used as `sender`.

```
1. The account which has been created already.
2. The account which has sufficient balance of currencies in items.
3. The account that you(or owners of the account) know its private keys corresponding␣
↪account public keys.
```

Then, create *fact*!

```
/*
import org.mitumc.sdk.operation.currency.*;
*/
String senderAddress = "CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca"; // sender's␣
↪account address; replace with your address
CreateAccountsFact fact = generator.mc().getCreateAccountsFact(senderAddress, new␣
↪CreateAccountsItem[]{ item });  // newCreateAccountsFact(sender address, item list)
```

If you want to create fact with multiple items, put them all in item list of
`Generator.mc().getCreateAccountsFact(sender's address, item list)` as an array.

Create Operation

Finally, you are in the step to create operation!

Only thing you need to prepare is **sender's private key**. It is used for signing fact.
The signature of a private key is included in `fact_signs` as a **fact signature**.
The sum of weights of all signers in `fact_signs` should exceed or be equal to the `sender`'s threshold.

Only the signatures of the sender account's keys are available to fact_signs!

There is `memo` in operation but it is not necessary. You can enter something if you need, but be careful because that `memo` also affects the `operation hash`.

In this example, suppose that `sender` is a *single-sig account* which means only a single key exists in the sender's account.
If `sender` is a *multi-sig account*, you may add multiple signatures to `fact_signs`.
What key must sign is decided by the account's threshold and keys' weights.

```
/*
import org.mitumc.sdk.operation.Operation;
*/
String senderPrivateKey = "KxD8T82nfwsUmQu3iMXENm93YTTatGFp1AYDPqTo5e6ycvY1xNXpmpr"; //␣
↪sender's private key; replace with your private key

Operation operation = generator.getOperation(fact); // getOperation(fact, memo); enter␣
↪memo if you need
operation.addSign(senderPrivateKey); // addSign(private key) add fact signature to fact_
↪signs
```

Use just `Generator.getOperation(fact, memo)` for create operations, not
`Generator.currency().newOperation(fact, memo)`.

Unfortunately, an operation can contain only one fact.

Create Seal

In fact, `operation` itself is enough to create an account.

However, sometimes you may need to wrap multiple operations with a seal.

As mentioned above, one seal can contain multiple operations.

The maximum number of operations in a seal is decided by the policy of nodes.
So check how many operations you can include in a seal before creating seals.

Anyway, it is simple to create a seal with **mitum-java-util**.

What you have to prepare is *private key* from Mitum key package without any conditions.
Any *btc compressed wif* with suffix *mpr* is okay.

```
String signKey = "KzafpyGojcN44yme25UMGvZvKWdMuFv1SwEhsZn8iF8szUz16jskmpr";
HashMap<String, Object> seal = gn.getSeal(signKey, new Operation[]{ operation }); //␣
↪getSeal(sign key, operation list)
```

Like `getOperation`, use `Generator.getSeal(signer, operation list)`.

Put all operations to wrap in *operation list*.

Support Operations

This section will introduce code example for each operation.

What Mitum Currency operations **mitum-java-util** supports are,

- Create Account
- Key Updater
- Transfer

Create Account

The tutorial for `create-account` have been already explained but it'll be re-introduced in one code-block.

To create a new account you have to prepare,

- The information of the new account: account keys as pairs of (public key, weight), threshold, initial balance as pairs of (currency id, amount).
- Sender's account that has existed already - especially sender's account address and private keys.

As mentioned before, what private keys must sign the fact is up to the threshold and composition of weights.

```
/*
import org.mitumc.sdk.key.*;
import org.mitumc.sdk.Generator;
import org.mitumc.sdk.operation.Operation;
import org.mitumc.sdk.operation.currency.*;
*/

String senderPrivateKey = "KzafpyGojcN44yme25UMGvZvKWdMuFv1SwEhsZn8iF8szUz16jskmpr";
String senderAddress = "FcLfoPNCYjSMnxLPiQJQFGTV15ecHn3xY4J2HNCrqbCfmca";

Generator gn = Generator.get("mitum"); // network id: mitum

Key key = gn.mc().key("knW2wVXH399P9Xg8aVjAGuMkk3uTBZwcSpcy4aR3UjiAmpu", 100);
Keys keys = gn.mc().keys(new Key[]{ key }, 100); // becomes single-sig account

Amount amount = gn.mc().amount("MCC", "1000");
CreateAccountsItem item = gn.mc().getCreateAccountsItem(keys, new Amount[]{ amount });

CreateAccountsFact fact = gn.mc().getCreateAccountsFact(sourceAddr, new␣
↪CreateAccountsItem[]{ item });

Operation createAccount = gn.getOperation(fact);
createAccount.addSign(senderPrivateKey);
```

The detailed explanation was omitted. Refer to the beginning part of 'Make Your First Operation.'.

Key Updater

This operation is to update keys of the account as its name implies.

For example,

```
- I have an single sig account with keys:␣
↪(kpYjRwq6gQrjvzeqQ91MNiCcR9Beb9sD67SuhQ6frPGwmpu, 100), threshold: 100
- But I want to replace keys of the account with keys:␣
↪(22ndFZw57ax28ydC3ZxzLJMNX9oMSqAfgauyWhC17pxDpmpu, 50),␣
↪(22wD5RWsRFAr8mHkYmmyUDzKf6VBNgjHcgc3YhKxCvrZDmpu, 50), threshold: 100
- Then you can use key-updater operation to reach the goal!
```

Can I change my account from single-sig to multi-sig? or from multi-sig to single-sig?

Fortunately, of course, you can!

To update keys of the account, you have to prepare,

- The account(target) information you want to change the keys - account address and private keys; what private keys need is up to threshold and key weights.
- New keys: pairs of (public key, weights) and threshold
- Sufficient balance in a currency id to pay a fee.

`create-account` and `transfer` need `item` to create an operation but `key-updater` don't need any item for it. Just create *fact* right now.

```java
/*
import org.mitumc.sdk.key.*;
import org.mitumc.sdk.Generator;
import org.mitumc.sdk.operation.Operation;
import org.mitumc.sdk.operation.currency.*;
*/

Generator gn = Generator.get("mitum"); // network id: mitum

String targetPrivateKey = "KzejtzpPZFdLUXo2hHouamwLoYoPtoffKo5zwoJXsBakKzSvTdbzmpr";
String targetAddress = "JDhSSB3CpRjwM8aF2XX23nTpauv9fLhxTjWsQRm9cJ7umca";

Key key1 = gn.mc().key("22ndFZw57ax28ydC3ZxzLJMNX9oMSqAfgauyWhC17pxDpmpu", 50);
Key key2 = gn.mc().key("22wD5RWsRFAr8mHkYmmyUDzKf6VBNgjHcgc3YhKxCvrZDmpu", 50);
Keys newKeys = gn.mc().keys(new Key[]{ key1, key2 }, 100);
```

(continues on next page)

(continued from previous page)

```
KeyUpdaterFact fact = gn.mc().getKeyUpdaterFact(sourceAddr, "MCC", newKeys); //␣
↪getKeyUpdaterFact(target address, currency for fee, new keys)
Operation keyUpdater = gn.getOperation(fact);
keyUpdater.addSign(targetPrivateKey);
```

- **After updating keys of the account, the keys used before become useless. You should sign operation with private keys of new keypairs of the account.**
- **So record new private keysthreshold somewhere else before sending a key-updater operation to the network.**

Transfer

Finally, you can transfer your tokens to another account.

As other operations, you have to prepare,

- Sender's account information - account address, and private keys
- Pairs of (currency id, amount) to transfer

Like `create-account`, you must create *item* before making *fact*.

Check whether you hold sufficient balance for each currency id to transfer before sending the operation.

Before start, suppose that you want to transfer,

- 1000000 MCC token
- 15000 PEN token

And the receiver is,

- CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca

Note that up to 10 (currency id, amount) pairs can be included in one item.
Moreover, up to 10 items can be included in one fact. However, the receiver for each item should be different.

```
/*
import org.mitumc.sdk.Generator;
import org.mitumc.sdk.operation.Operation;
import org.mitumc.sdk.operation.currency.*;
*/
```

(continues on next page)

(continued from previous page)

```
Generator gn = Generator.get("mitum"); // network id: mitum

String senderPrivateKey = "KzdeJMr8e2fbquuZwr9SEd9e1ZWGmZEj96NuAwHnz7jnfJ7FqHQBmpr";
String senderAddress = "2D5vAb2X3Rs6ZKPjVsK6UHcnGxGfUuXDR1ED1hcvUHqsmca";
String receiverAddress = "CY1pkxsqQK6XMbnK4ssDNbDR2K7mitSwdS27DwBjd3Gcmca";

Amount amount1 = gn.mc().amount("1000000", "MCC")
Amount amount2 = gn.mc().amount("15000", "PEN")

TransfersItem item = gn.mc().getTransfersItem(receiverAddress, new Amount[]{ amount1,␣
↪amount2 }); // getTransfersItem(receiver address, amount list)
TransfersFact fact = gn.mc().getTransfersFact(sourceAddr, new TransfersItem[]{ item }); /
↪/ getTransfersFact(sender address, item list)

Operation transfer = gn.getOperation(fact);
transfer.addSign(senderPrivateKey); // suppose sender is single-sig
```

There are other operations that **mitum-java-util** supports, like operations of *Mitum Document, but this document doesn't provide examples of those operations.
Refer to README if necessary.

2.21.3 Sign

To allow an operation to be stored in blocks, whether signatures of the operation satisfy the **condition** should be checked.

What you have to care about is,

- Has every signature been signed by the private key of the account?
- Is the sum of every weight for each signer greater than or equal to the account threshold?

Of course, there are other conditions each operation must satisfy but we will focus on **signature** (especially about fact signature) in this section.

Let's suppose there is a multi-sig account with 3 keys s.t each weight is 30 and threshold is 50.

That means,

- (pub1, 30)
- (pub2, 30)
- (pub3, 30)
- threshold: 50

When this account wants to send an operation, the operation should include at least two fact signatures of different signers.

1. CASE1: fact signatures signed by pub1's private key and pub2's private key
    1. the sum of pub1's weight and pub2's weight: 60
    2. the sum of weights = 60 > threshold = 50
    3. So the operation with these two fact signatures is available
2. CASE2: fact signatures signed by pub2's private key and pub3's private key
    1. the sum of pub2's weight and pub3's weight: 60
    2. the sum of weights = 60 > threshold = 50
    3. So the operation with these two fact signatures is available
3. CASE3: fact signatures signed by pub1's private key and pub3's private key
    1. the sum of pub1's weight and pub3's weight: 60
    2. the sum of weights = 60 > threshold = 50
    3. So the operation with these two fact signatures is available
4. CASE4: fact signatures signed by pub1's private key, pub2's private key, pub3's private key
    1. the sum of pub1's weight, pub2's weight and pub3's weight: 90
    2. the sum of weights = 90 > threshold = 50
    3. So the operation with these two fact signatures is available

Therefore, you must add multiple signatures to each operation to satisfy the condition. (use `Operation.addSign(private key)`)
Like **CASE4**, it's okay to sign with every private key as long as the sum of their weight >= threshold.

Add Fact Sign to Operation

Besides adding a fact signature when creating the operation, there is another way to add a new fact signature to the operation.

To add a new signature to the operation, you have to prepare,

- Private key to sign - it should be that of the sender of the operation.
- Operation as JsonObject, or external JSON file.
- Network ID

First, create `Signer` with `network id` like `Generator`.

```
/*
import org.mitumc.sdk.Signer;
import org.mitumc.sdk.JSONParser;
*/
String id = "mitum";
String key = "KzafpyGojcN44yme25UMGvZvKWdMuFv1SwEhsZn8iF8szUz16jskmpr";

Signer signer = Signer.get(id, key);
```

Then, sign now!

```
HashMap<String, Object> signed = signer.addSignToOperation("operation.json"); // or␣
↪JsonObject from Operation JSON instead
```

Note that the result operation is not `Operation` object of **mitum-java-util**. It's just a HashMap object.
If you want to add multiple signatures at once, you must create a separate JSON file then re-sign it with other private keys using `Signer`.

2.21.4 Details

Get Mitum Keypair

We will introduce how to create Mitum keypairs!

Before start, we want to let you know something important; About type suffix.

Address, *private key*, and *public key* in Mitum have specific type suffixes. They are,

- Account Address: `mca`
- Private Key: `mpr`
- Public Key: `mpu`

For example, an single-sig account looks like,

- Account Address: `9XyYKpjad2MSPxR4wfQHvdWrZnk9f5s2zc9Rkdy2KT1gmca`
- Private Key: `L11mKUECzKouwvXwh3eyECsCnvQx5REureuujGBjRuYXbMswFkMxmpr`
- Public Key: `28Hhy6jwkEHx75bNLmG66RQu1LWiZ1vodwRTURtBJhtPWmpu`

There are three methods to create a keypair.

Just Create New Keypair

mitum-java-util will create a random keypair for you!

Use `Keypar.create()`.

```
/*
import org.mitumc.sdk.key.Keypair;
*/
Keypair kp = Keypair.create();

kp.getPrivateKey(); // returns private key of the keypair
kp.getPublicKey(); // returns public key of the keypair
```

Get Keypair From Your Private Key

If you already have your own private key, create keypair with it!

```
/*
import org.mitumc.sdk.key.Keypair;
*/
String key = "KzafpyGojcN44yme25UMGvZvKWdMuFv1SwEhsZn8iF8szUz16jskmpr";
Keypair pkp = Keypair.fromPrivateKey(key);
```

Get Keypair From Your Seed

You can get a keypair from your seed, too. Even if you don't remember the private key of the keypair, the keypair can be recovered by its seed.
Note that string seed length >= 36.

```
/*
import org.mitumc.sdk.key.Keypair;
*/
String seed =  "Thisisaseedfortheexample;Keypair.fromSeed()";
Keypair skp = Keypair.fromSeed(seed);

// or... -----------------------------//
// byte[] bseed = seed.getBytes();
// Keypair skp = Keypair.fromSeed(bseed);
```

Get Account Address with Keys

You can calculate address from threshold, and every (public key, weight) pair of the account.

However, it is not available to get an address if the keys or threshold of the account have changed.
This method is available only for the account that have not changed yet.

The account information for the example is,

- key1: (vmk1iprMrs8V1NkA9DsSL3XQNnUW9SmFL5RCVJC24oFYmpu, 40)
- key2: (29BQ8gcVfJd5hPZCKj335WSe4cyDe7TGrjam7fTrkYNunmpu, 30)
- key3: (uJKiGLBeXF3BdaDMzKSqJ4g7L5kAukJJtW3uuMaP1NLumpu, 30)
- threshold: 100

```
/*
import org.mitumc.sdk.Generator
import org.mitumc.key.Key
import org.mitumc.key.Keys
*/
Generator generator = Generator.get("mitum");

Key key1 = generator.mc().key("vmk1iprMrs8V1NkA9DsSL3XQNnUW9SmFL5RCVJC24oFYmpu", 40);
Key key2 = generator.mc().key("29BQ8gcVfJd5hPZCKj335WSe4cyDe7TGrjam7fTrkYNunmpu", 30);
Key key3 = generator.mc().key("uJKiGLBeXF3BdaDMzKSqJ4g7L5kAukJJtW3uuMaP1NLumpu", 30);

Keys keys = generator.currency().keys(new Key[]{ key1, key2, key3 }, 100);

String address = keys.getAddress(); // This is the goal!
```

Major Classes

Generator

`Generator` is the class that helps generate operations for Mitum Currency.

Before you use `Generator`, `network id` must be set.

- For **Mitum Currency**, use `Generator.mc()`.
- For **Mitum Document**, use `Generator.md()`.

For details of generating operations for **Mitum Document**, refer to README.

```
/*
import org.mitumc.sdk.Generator;
*/
String id = "mitum";
Generator generator = Generator.get(id);

CurrencyGenerator cgn = generator.mc(); // org.mitumc.sdk.operation.currency.
↪CurrencyGenerator;
DocumentGenerator bgn = generator.md(); // org.mitumc.sdk.operation.document.
↪DocumentGenerator;
```

All methods of `Generator` provides are,

```
/* For Mitum Currency */
Generator.mc().key(String key, int weight);
Generator.mc().keys(Key[] keys, int threshold);
Generator.mc().amount(String currency, String amount);
Generator.mc().getCreateAccountsItem(Keys keys, Amount[] amounts);
Generator.mc().getTransfersItem(String receiver, Amount[] amounts);
Generator.mc().getCreateAccountsFact(String sender, CreateAccountsItem[] items);
Generator.mc().getKeyUpdaterFact(String target, String currencyId, Keys keys);
Generator.mc().getTransfersFact(String sender, TransfersItem[] items);

/* For Mitum Document */
Generator.md().getCreateDocumentsItem(Document document, String currencyId);
Generator.md().getUpdateDocumentsItem(Document document, String currencyId);
Generator.md().getCreateDocumentsFact(String sender, CreateDocumentsItem[] items);
Generator.md().getUpdateDocumentsFact(String sender, UpdateDocumentsItem[] items);

/* For Blocksign */
Generator.md().bs().user(String address, String signCode, boolean signed);
Generator.md().bs().document(String documentId, String owner, String fileHash,␣
↪BlockSignUser creator, String title, String size, BlockSignUser[] signers);
Generator.md().bs().getSignDocumentsItem(String documentId, String owner, String␣
↪currencyId);
Generator.md().bs().getSignDocumentsFact(String sender, SignDocumentsItem[] items);
↪DocumentsFact(String sender, BlockCityItem<T>[] items);

/* For Blockcity */
Generator.md().bc().candidate(String address, String nickname, String manifest, int␣
↪count);
Generator.md().bc().userStatistics(int hp, int strength, int agility, int dexterity, int␣
↪charisma, int intelligence, int vital);
Generator.md().bc().document(String documentId, String owner, int gold, int bankGold,␣
↪UserStatistics statistics);
Generator.md().bc().document(String documentId, String owner, String address, String␣
↪area, String renter, String account, String rentDate, int period);
Generator.md().bc().document(String documentId, String owner, int round, String endTime,␣
↪Candidate[] candidates, String bossName, String account, String office);
Generator.md().bc().document(String documentId, String owner, String name, String␣
```

(continues on next page)

(continued from previous page)

```
↪account, String date, String usage, String app);

/* Common */
Generator.getOperation(OperationFact fact);
Generator.getOperation(String memo, OperationFact fact);
Generator.getSeal(String signKey, Operation[] operations);
```

Signer

`Signer` is the class for adding new fact signature to already create operations.

Like `Generator`, `network id` must be set.

You have to prepare *private key* to sign, too.

`Signer` provides only one method, that is,

```
HashMap<String, Object> addSignToOperation(JsonObject operation);
HashMap<String, Object> addSignToOperation(String operationPath);
```

To check the exact usage of `Signer`, go back to **Make Your First Operation - Sign**.

JSONParser

This class is constructed just for convenience.
If you would like to use other Java packages to export `Operation` to file or to print it in JSON format, you don't need to use `JSONParser` of **mitum-java-util**.

```
/*
import org.mitumc.sdk.JSONParser;
*/
// ... omitted
// ... create operations
// ... refer to above `Make Your First Operation`
// ... suppose you have already made operations - createAccount, keyUpdater, transfer␣
↪and a seal - seal

JSONParser.createJSON(createAccount.toDict(), 'createAccount.json'); //␣
↪createJSON(HashMap, filePath)
JSONParser.createJSON(keyUpdater.toDict(), 'keyUpdater.json');
JSONParser.createJSON(transfer.toDict(), 'transfer.json');
JSONParser.createJSON(seal, 'seal.json');
```

2.22 About Time Stamp

2.22.1 Expression of Time Stamp

For blocks, seals, signatures and etc, mitum uses `yyyy-MM-dd HH:mm:ss.* +0000 UTC` expression and `yyyy-MM-ddTHH:mm:ss.*Z` as standard.
All other timezones are not allowed! You must use only `+0000 timezone` for mitum.

For example,

- When converting timestamp to byte format for generating block/seal/fact_sign hash

```
convert the string 2021-11-16 01:53:30.518 +0000 UTC to bytes format
```

- When putting timestamp in block, seal, fact_sign or etc

```
convert the timestamp to 2021-11-16T01:53:30.518Z and put it in json
```

To generate operation hash, mitum concatenates byte arrays of network id, fact hash and byte arrays of fact_signs.
And to generate the byte array of a fact_sign, mitum concatenates byte arrays of signer, signature digest and signed_at.

Be careful that the format of signed_at when converted to bytes is like `yyyy-MM-dd HH:mm:ss.* +0000 UTC` but it will be expressed as `yyyy-MM-ddTHH:mm:ss.*Z` when putted in json.

2.22.2 How many decimal places to be expressed?

There is one more thing to note.

First at all, you don't have to care about decimal points of second(ss.*) in timestamp.
Moreover, you can write timestamp without . and any number under ..
However, you should not put any unnecessary zeros(0) in the float expression of second(ss.*) when converting timestamp to bytes format.

For example,

- `2021-11-16T01:53:30.518Z` is converted to `2021-11-16 01:53:30.518 +0000 UTC` without any change of the time itself.
- `2021-11-16T01:53:30.510Z` must be converted to `2021-11-16 01:53:30.51 +0000 UTC` when generating hash.
- `2021-11-16T01:53:30.000Z` must be converted to `2021-11-16T01:53:30 +0000 UTC` when generating hash.

Any timestamp with some unnecessary zeros putted in json doesn't affect to effectiveness of the block, seal, or operation.

Just pay attention when convert the format.