Hyperledger Indy Node Documentation

Hyperledger

Aug 07, 2019

Contents

CHAPTER 1

Transactions

1.1 General Information

This doc is about supported transactions and their representation on the Ledger (that is, the internal one). If you are interested in the format of a client's request (both write and read), then have a look at *requests*.

- All transactions are stored in a distributed ledger (replicated on all nodes)
- The ledger is based on a Merkle Tree
- The ledger consists of two things:
  - transactions log as a sequence of key-value pairs where key is a sequence number of the transaction and value is the serialized transaction
  - merkle tree (where hashes for leaves and nodes are persisted)
- Each transaction has a sequence number (no gaps) - keys in transactions log
- So, this can be considered a blockchain where each block's size is equal to 1
- There are multiple ledgers by default:
  - *pool ledger*: transactions related to pool/network configuration (listing all nodes, their keys and addresses)
  - *config ledger*: transactions for pool configuration plus transactions related to pool upgrade
  - *domain ledger*: all main domain and application specific transactions (including NYM transactions for DID)
- All transactions are serialized to MsgPack format
- All transactions (both transaction log and merkle tree hash stores) are stored in a LevelDB
- One can use the `read_ledger` script to get transactions for a specified ledger in a readable format (JSON)
- See *roles and permissions* for a list of roles and they type of transactions they can create.

Below you can find the format and description of all supported transactions.

1.2 Genesis Transactions

As Indy is a public **permissioned** blockchain, each ledger may have a number of pre-defined transactions defining the initial pool and network.

- pool genesis transactions define initial trusted nodes in the pool
- domain genesis transactions define initial trusted trustees and stewards

1.3 Common Structure

Each transaction has the following structure consisting of metadata values (common for all transaction types) and transaction specific data:

```
{
    "ver": <...>,
    "txn": {
        "type": <...>,
        "protocolVersion": <...>,
```

(continues on next page)

(continued from previous page)

```
        "data": {
            "ver": <...>,
            <txn-specific fields>
        },

        "metadata": {
            "reqId": <...>,
            "from": <...>,
            "endorser": <...>,
            "digest": <...>,
            "payloadDigest": <...>,
            "taaAcceptance": {
                "taaDigest": <...>,
                "mechanism": <...>,
                "time": <...>
             }
        },
    },
    "txnMetadata": {
        "txnTime": <...>,
        "seqNo": <...>,
        "txnId": <...>
    },
    "reqSignature": {
        "type": <...>,
        "values": [{
            "from": <...>,
            "value": <...>
        }]
    }
}
```

- `ver` (string):

  Transaction version to be able to evolve content. The content of all sub-fields may depend on this version.

- `txn` (dict):

  Transaction-specific payload (data)

  – `type` (enum number as string):

    Supported transaction types:

    * NODE = “0”
    * NYM = “1”
    * TXN_AUTHOR_AGREEMENT = “4”
    * TXN_AUTHOR_AGREEMENT_AML = “5”
    * ATTRIB = “100”
    * SCHEMA = “101”
    * CLAIM_DEF = “102”
    * POOL_UPGRADE = “109”
    * NODE_UPGRADE = “110”
    * POOL_CONFIG = “111”

    * REVOC_REG_DEF = "113"
    * REVOC_REG_DEF = "114"
    * AUTH_RULE = "120"
    * AUTH_RULES = "122"

- `protocolVersion` (integer; optional):

  The version of client-to-node or node-to-node protocol. Each new version may introduce a new feature in requests/replies/data. Since clients and different nodes may be at different versions, we need this field to support backward compatibility between clients and nodes.

- `data` (dict):

  Transaction-specific data fields (see following sections for each transaction's description).

- `metadata` (dict):

  Metadata as came from the request.

  * `from` (base58-encoded string):

    Identifier (DID) of the transaction author as base58-encoded string for 16 or 32 bit DID value. It may differ from `endorser` field who submits the transaction on behalf of `identifier`. If `endorser` is absent, then the author (`identifier`) plays the role of endorser and submits request by his own. It also may differ from `dest` field for some of requests (for example NYM), where `dest` is a target identifier (for example, a newly created DID identifier).

    Example:

    · `identifier` is a DID of a transaction author who doesn't have write permissions; `endorser` is a DID of a user with Endorser role (that is with write permissions).

    · new NYM creation: `identifier` is a DID of an Endorser creating a new DID, and `dest` is a newly created DID.

  * `reqId` (integer): Unique ID number of the request with transaction.

  * `digest` (SHA256 hex digest string): SHA256 hash hex digest of all fields in the initial requests (including signatures)

  * `payloadDigest` (SHA256 hex digest string): SHA256 hash hex digest of the payload fields in the initial requests, that is all fields excluding signatures and plugins-added ones

  * `endorser` (base58-encoded string, optional): Identifier (DID) of an Endorser submitting a transaction on behalf of the original author (`identifier`) as base58-encoded string for 16 or 32 bit DID value. If `endorser` is absent, then the author (`identifier`) plays the role of endorser and submits request by his own. If `endorser` is present then the transaction must be multi-signed by the both author (`identifier`) and Endorser (`endorser`).

  * `taaAcceptance` (dict, optional): If transaction author agreement is set/enabled, then every transaction (write request) from Domain and plugins-added ledgers must include acceptance of the latest transaction author agreement.

    · `taaDigest` (SHA256 hex digest string): SHA256 hex digest of the latest Transaction Author Agreement on the ledger. The digest is calculated from concatenation of *TRANSACTION_AUTHOR_AGREEMENT*'s `version` and `text`.

    · `mechanism` (string): a mechanism used to accept the signature; must be present in the latest list of transaction author agreement acceptane mechanisms on the ledger

    · `time` (integer as POSIX timestamp): transaction author agreement acceptance time

  - `txnMetadata` (dict):

    Metadata attached to the transaction.

    * `version` (integer): Transaction version to be able to evolve `txnMetadata`. The content of `txnMetadata` may depend on the version.
    * `txnTime` (integer as POSIX timestamp): The time when transaction was written to the Ledger as POSIX timestamp.
    * `seqNo` (integer): A unique sequence number of the transaction on Ledger
    * `txnId` (string, optional): Txn ID as State Trie key (address or descriptive data). It must be unique within the ledger. Usually present for Domain transactions only.

- `reqSignature` (dict):

  Submitter's signature over request with transaction (`txn` field).

  - `type` (string enum):
    * ED25519: ed25519 signature
    * ED25519_MULTI: ed25519 signature in multisig case.
  - `values` (list):
    * `from` (base58-encoded string): Identifier (DID) of signer as base58-encoded string for 16 or 32 byte DID value.
    * `value` (base58-encoded string): signature value

Please note that all these metadata fields may be absent for genesis transactions.

1.4 Domain Ledger

1.4.1 NYM

Creates a new NYM record for a specific user, endorser, steward or trustee. Note that only trustees and stewards can create new endorsers and a trustee can be created only by other trustees (see *roles*).

The transaction can be used for creation of new DIDs, setting and rotation of verification key, setting and changing of roles.

- `dest` (base58-encoded string):

  Target DID as base58-encoded string for 16 or 32 byte DID value. It may differ from the `from` metadata field, where `from` is the DID of the submitter. If they are equal (in permissionless case), then transaction must be signed by the newly created `verkey`.

  Example: `from` is a DID of a Endorser creating a new DID, and `dest` is a newly created DID.

- `role` (enum number as integer; optional):

  Role of a user that the NYM record is being created for. One of the following values

  - None (common USER)
  - "0" (TRUSTEE)
  - "2" (STEWARD)
  - "101" (ENDORSER)

    - "201" (NETWORK_MONITOR)

  A TRUSTEE can change any Nym's role to None, thus stopping it from making any further writes (see *roles*).

- `verkey` (base58-encoded string, possibly starting with "~"; optional):

  Target verification key as base58-encoded string. It can start with "~", which means that it's an abbreviated verkey and should be 16 bytes long when decoded, otherwise it's a full verkey which should be 32 bytes long when decoded. If not set, then either the target identifier (`did`) is 32-bit cryptonym CID (this is deprecated), or this is a user under guardianship (doesn't own the identifier yet). Verkey can be changed to "None" by owner, it means that this user goes back under guardianship.

- `alias` (string; optional):

  NYM's alias.

If there is no NYM transaction for the specified DID (`did`) yet, then this can be considered as the creation of a new DID.

If there is already a NYM transaction with the specified DID (`did`), then this is is considered an update of that DID. In this case **only the values that need to be updated should be specified** since any specified one is treated as an update even if it matches the current value in ledger. All unspecified values remain unchanged.

So, if key rotation needs to be performed, the owner of the DID needs to send a NYM request with `did` and `verkey` only. `role` and `alias` will stay the same.

Example:

```
{
    "ver": 1,
    "txn": {
        "type":"1",
        "protocolVersion":2,

        "data": {
            "ver": 1,
            "dest":"GEzcdDLhCpGCYRHW82kjHd",
            "verkey":"~HmUWn928bnFT6Ephf65YXv",
            "role":101,
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
            "taaAcceptance": {
                "taaDigest":
↪"6sh15d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
                "mechanism": "EULA",
                "time": 1513942017
             }
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
        "txnId": "N22KY2Dyvmuu2PyyqSFKue|01"
    },
```

(continues on next page)

(continued from previous page)

```
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }

}
```

1.4.2 ATTRIB

Adds an attribute to a NYM record

- `dest` (base58-encoded string):

  Target DID we set an attribute for as base58-encoded string for 16 or 32 byte DID value. It differs from `from` metadata field, where `from` is the DID of the submitter.

  Example: `from` is a DID of a Endorser setting an attribute for a DID, and `dest` is the DID we set an attribute for.

- `raw` (sha256 hash string; mutually exclusive with `hash` and `enc`):

  Hash of the raw attribute data. Raw data is represented as JSON, where the key is the attribute name and the value is the attribute value. The ledger only stores a hash of the raw data; the real (unhashed) raw data is stored in a separate attribute store.

- `hash` (sha256 hash string; mutually exclusive with `raw` and `enc`):

  Hash of attribute data (as sent by the client). The ledger stores this hash; nothing is stored in an attribute store.

- `enc` (sha256 hash string; mutually exclusive with `raw` and `hash`):

  Hash of encrypted attribute data. The ledger contains the hash only; the real encrypted data is stored in a separate attribute store.

Example:

```
{
    "ver": 1,
    "txn": {
        "type":"100",
        "protocolVersion":2,

        "data": {
            "ver":1,
            "dest":"GEzcdDLhCpGCYRHW82kjHd",
            "raw":"3cba1e3cf23c8ce24b7e08171d823fbd9a4929aafd9f27516e30699d3a42026a",
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
```

(continues on next page)

(continued from previous page)

```
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
            "taaAcceptance": {
                "taaDigest":
↪"6sh15d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
                "mechanism": "EULA",
                "time": 1513942017
             }
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
        "txnId": "N22KY2Dyvmuu2PyyqSFKue|02"
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.4.3 SCHEMA

Adds a Claim's schema.

It's not possible to update an existing schema. So, if the Schema needs to be evolved, a new Schema with a new version or new name needs to be created.

- `data` (dict):

  Dictionary with Schema's data:

  – `attr_names`: array of attribute name strings

  – `name`: Schema's name string

  – `version`: Schema's version string

Example:

```
{
    "ver": 1,
    "txn": {
        "type":101,
        "protocolVersion":2,

        "data": {
            "ver":1,
            "data": {
                "attr_names": ["undergrad","last_name","first_name","birth_date",
↪"postgrad","expiry_date"],
                "name":"Degree",
```

(continues on next page)

(continued from previous page)

```
                "version":"1.0"
            },
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "endorser": "D6HG5g65TDQr1PPHHRoiGf",
            "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
            "taaAcceptance": {
                "taaDigest":
↪"6sh15d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
                "mechanism": "EULA",
                "time": 1513942017
             }
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
        "txnId":"L5AD5g65TDQr1PPHHRoiGf1|Degree|1.0",
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }

}
```

1.4.4 CLAIM_DEF

Adds a claim definition (in particular, public key), that Issuer creates and publishes for a particular claim schema.

- `data` (dict):

  Dictionary with claim definition's data:

  - `primary` (dict): primary claim public key
  - `revocation` (dict): revocation claim public key

- `ref` (string):

  Sequence number of a schema transaction the claim definition is created for.

- `signature_type` (string):

  Type of the claim definition (that is claim signature). `CL` (Camenisch-Lysyanskaya) is the only supported type now.

- `tag` (string, optional):

A unique tag to have multiple public keys for the same Schema and type issued by the same DID. A default tag `tag` will be used if not specified.

Example:

```
{
    "ver": 1,
    "txn": {
        "type":102,
        "protocolVersion":2,

        "data": {
            "ver":1,
            "data": {
                "primary": {
                    ...
                },
                "revocation": {
                    ...
                }
            },
            "ref":12,
            "signature_type":"CL",
            'tag': 'some_tag'
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "endorser": "D6HG5g65TDQr1PPHHRoiGf",
            "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
            "taaAcceptance": {
                "taaDigest":
↪"6sh15d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
                "mechanism": "EULA",
                "time": 1513942017
             }
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
        "txnId":"HHAD5g65TDQr1PPHHRoiGf2L5AD5g65TDQr1PPHHRoiGf1|Degree1|CL|key1",
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.4.5 REVOC_REG_DEF

Adds a Revocation Registry Definition, that Issuer creates and publishes for a particular Claim Definition. It contains public keys, maximum number of credentials the registry may contain, reference to the Claim Def, plus some revocation registry specific data.

- `value` (dict):

  Dictionary with revocation registry definition's data:

  – `maxCredNum` (integer): a maximum number of credentials the Revocation Registry can handle

  – `tailsHash` (string): tails' file digest

  – `tailsLocation` (string): tails' file location (URL)

  – `issuanceType` (string enum): defines credentials revocation strategy. Can have the following values:

    * `ISSUANCE_BY_DEFAULT`: all credentials are assumed to be issued initially, so that Revocation Registry needs to be updated (REVOC_REG_ENTRY txn sent) only when revoking. Revocation Registry stores only revoked credentials indices in this case. Recommended to use if expected number of revocation actions is less than expected number of issuance actions.

    * `ISSUANCE_ON_DEMAND`: no credentials are issued initially, so that Revocation Registry needs to be updated (REVOC_REG_ENTRY txn sent) on every issuance and revocation. Revocation Registry stores only issued credentials indices in this case. Recommended to use if expected number of issuance actions is less than expected number of revocation actions.

  – `publicKeys` (dict): Revocation Registry's public key

- `id` (string): Revocation Registry Definition's unique identifier (a key from state trie is currently used)
- `credDefId` (string): The corresponding Credential Definition's unique identifier (a key from state trie is currently used)
- `revocDefType` (string enum): Revocation Type. `CL_ACCUM` (Camenisch-Lysyanskaya Accumulator) is the only supported type now.
- `tag` (string): A unique tag to have multiple Revocation Registry Definitions for the same Credential Definition and type issued by the same DID.

Example:

```
{
    "ver": 1,
    "txn": {
        "type":113,
        "protocolVersion":2,

        "data": {
            "ver":1,
            'id': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_
↪ACCUM:tag1',
            'credDefId': 'FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag'
            'revocDefType': 'CL_ACCUM',
            'tag': 'tag1',
            'value': {
                'maxCredNum': 1000000,
                'tailsHash': '6619ad3cf7e02fc29931a5cdc7bb70ba4b9283bda3badae297',
                'tailsLocation': 'http://tails.location.com',
                'issuanceType': 'ISSUANCE_BY_DEFAULT',
                'publicKeys': {},
```

(continues on next page)

(continued from previous page)

```
            },
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "endorser": "D6HG5g65TDQr1PPHHRoiGf",
            'digest':
↪'4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453',
            'payloadDigest':
↪'21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685',
            "taaAcceptance": {
                "taaDigest":
↪"6sh15d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
                "mechanism": "EULA",
                "time": 1513942017
             }
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
        "txnId":"L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_
↪ACCUM:tag1",
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.4.6 REVOC_REG_ENTRY

The RevocReg entry containing the new accumulator value and issued/revoked indices. This is just a delta of indices, not the whole list. So, it can be sent each time a new claim is issued/revoked.

- `value` (dict):

  Dictionary with revocation registry's data:

  - `accum` (string): the current accumulator value
  - `prevAccum` (string): the previous accumulator value; it's compared with the current value, and txn is rejected if they don't match; it's needed to avoid dirty writes and updates of accumulator.
  - `issued` (list of integers): an array of issued indices (may be absent/empty if the type is ISSUANCE_BY_DEFAULT); this is delta; will be accumulated in state.
  - `revoked` (list of integers): an array of revoked indices (delta; will be accumulated in state)

- `revocRegDefId` (string): The corresponding Revocation Registry Definition's unique identifier (a key from state trie is currently used)

- `revocDefType` (string enum): Revocation Type. `CL_ACCUM` (Camenisch-Lysyanskaya Accumulator) is the only supported type now.

Example:

```
{
    "ver": 1,
    "txn": {
        "type":114,
        "protocolVersion":2,

        "data": {
            "ver":1,
            'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1'
            'revocDefType': 'CL_ACCUM',
            'value': {
                'accum': 'accum_value',
                'prevAccum': 'prev_acuum_value',
                'issued': [],
                'revoked': [10, 36, 3478],
            },
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "endorser": "D6HG5g65TDQr1PPHHRoiGf",
            'digest':
↪'4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453',
            'payloadDigest':
↪'21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685',
            "taaAcceptance": {
                "taaDigest":
↪"6sh15d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
                "mechanism": "EULA",
                "time": 1513942017
             }
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
        "txnId":"5:L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_
↪ACCUM:tag1",
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.5 Pool Ledger

1.5.1 NODE

Adds a new node to the pool or updates an existing node in the pool

- `data` (dict):

  Data associated with the Node:

  - `alias` (string): Node's alias
  - `blskey` (base58-encoded string; optional): BLS multi-signature key as base58-encoded string (it's needed for BLS signatures and state proofs support)
  - `client_ip` (string; optional): Node's client listener IP address, that is the IP clients use to connect to the node when sending read and write requests (ZMQ with TCP)
  - `client_port` (string; optional): Node's client listener port, that is the port clients use to connect to the node when sending read and write requests (ZMQ with TCP)
  - `node_ip` (string; optional): The IP address other Nodes use to communicate with this Node; no clients are allowed here (ZMQ with TCP)
  - `node_port` (string; optional): The port other Nodes use to communicate with this Node; no clients are allowed here (ZMQ with TCP)
  - `services` (array of strings; optional): the service of the Node. `VALIDATOR` is the only supported one now.

- `dest` (base58-encoded string):

  Target Node's verkey as base58-encoded string for 16 or 32 byte DID value. It differs from `identifier` metadata field, where `identifier` is the DID of the transaction submitter (Steward's DID).

  Example: `identifier` is a DID of a Steward creating a new Node, and `dest` is the verkey of this Node.

If there is no NODE transaction with the specified Node ID (`dest`), then it can be considered as creation of a new NODE.

If there is a NODE transaction with the specified Node ID (`dest`), then this is update of existing NODE. In this case we can specify only the values we would like to override. All unspecified values remain the same. So, if a Steward wants to rotate BLS key, then it's sufficient to send a NODE transaction with `dest` and a new `blskey`. There is no need to specify all other fields, and they will remain the same.

Example:

```
{
    "ver": 1,
    "txn": {
        "type":0,
        "protocolVersion":2,

        "data": {
            "data": {
                "alias":"Delta",
                "blskey":
↪"4kkk7y7NQVzcfvY4SAe1HBMYnFohAJ2ygLeJd3nC77SFv2mJAmebH3BGbrGPHamLZMAFWQJNHEM81P62RfZjnb5SER6cQk1MNN
↪",
                "client_ip":"127.0.0.1",
                "client_port":7407,
```

(continues on next page)

(continued from previous page)

```
                "node_ip":"127.0.0.1",
                "node_port":7406,
                "services":["VALIDATOR"]
            },
            "dest":"4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2",
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.6 Config Ledger

1.6.1 POOL_UPGRADE

Command to upgrade the Pool (sent by Trustee). It upgrades the specified Nodes (either all nodes in the Pool, or some specific ones).

- `name` (string):

  Human-readable name for the upgrade.

- `action` (enum: `start` or `cancel`):

  Starts or cancels the Upgrade.

- `version` (string):

  The version of indy-node package we perform upgrade to. Must be greater than existing one (or equal if `reinstall` flag is True).

- `schedule` (dict of node DIDs to timestamps):

  Schedule of when to perform upgrade on each node. This is a map where Node DIDs are keys, and upgrade time is a value (see example below). If `force` flag is False, then it's required that time difference between each Upgrade must be not less than 5 minutes (to give each Node enough time and not make the whole Pool go down during Upgrade).

- `sha256` (sha256 hash string):

  sha256 hash of the package

- `force` (boolean; optional):

  Whether we should apply transaction (schedule Upgrade) without waiting for consensus of this transaction. If false, then transaction is applied only after it's written to the ledger. Otherwise it's applied regardless of result of consensus, and there are no restrictions on the Upgrade `schedule` for each Node. So, we can Upgrade the whole Pool at the same time when it's set to True. False by default. Avoid setting to True without good reason.

- `reinstall` (boolean; optional):

  Whether it's allowed to re-install the same version. False by default.

- `timeout` (integer; optional):

  Limits upgrade time on each Node.

- `justification` (string; optional):

  Optional justification string for this particular Upgrade.

Example:

```
{
    "ver": 1,
    "txn": {
        "type":109,
        "protocolVersion":2,

        "data": {
            "ver":1,
            "name":"upgrade-13",
            "action":"start",
            "version":"1.3",
            "schedule":{"4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2":"2017-12-
↪25T10:25:58.271857+00:00","AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3":"2017-12-
↪25T10:26:16.271857+00:00","DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2":"2017-12-
↪25T10:26:25.271857+00:00","JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ":"2017-12-
↪25T10:26:07.271857+00:00"},
            "sha256":"db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55
↪",
            "force":false,
            "reinstall":false,
            "timeout":1,
            "justification":null,
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
```

(continues on next page)

(continued from previous page)

```
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.6.2 NODE_UPGRADE

Status of each Node's upgrade (sent by each upgraded Node)

- `action` (enum string):

  One of `in_progress`, `complete` or `fail`.

- `version` (string):

  The version of indy-node the node was upgraded to.

Example:

```
{
    "ver":1,
    "txn": {
        "type":110,
        "protocolVersion":2,

        "data": {
            "ver":1,
            "action":"complete",
            "version":"1.2"
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
```

(continues on next page)

(continued from previous page)

```
        }]
    }
}
```

1.6.3 POOL_CONFIG

Command to change Pool's configuration

- `writes` (boolean):

  Whether any write requests can be processed by the pool (if false, then pool goes to read-only state). True by default.

- `force` (boolean; optional):

  Whether we should apply transaction (for example, move pool to read-only state) without waiting for consensus of this transaction. If false, then transaction is applied only after it's written to the ledger. Otherwise it's applied regardless of result of consensus. False by default. Avoid setting to True without good reason.

Example:

```
{
    "ver":1,
    "txn": {
        "type":111,
        "protocolVersion":2,

        "data": {
            "ver":1,
            "writes":false,
            "force":true,
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.6.4 AUTH_RULE

A command to change authentication rules. Internally authentication rules are stored as a key-value dictionary: `{action} -> {auth_constraint}`.

The list of actions is static and can be found in *auth_rules.md*. There is a default Auth Constraint for every action (defined in *auth_rules.md*).

The `AUTH_RULE` command allows to change the Auth Constraint. So, it's not possible to register new actions by this command. But it's possible to override authentication constraints (values) for the given action.

Please note, that list elements of `GET_AUTH_RULE` output can be used as an input (with a required changes) for `AUTH_RULE`.

The following input parameters must match an auth rule from the *auth_rules.md*:

- `auth_type` (string enum)

  The type of transaction to change the auth constraints to. (Example: "0", "1", ...). See transactions description to find the txn type enum value.

- `auth_action` (enum: `ADD` or `EDIT`)

  Whether this is adding of a new transaction, or editing of an existing one.

- `field` (string)

  Set the auth constraint of editing the given specific field. `*` can be used to specify that an auth rule is applied to all fields.

- `old_value` (string; optional)

  Old value of a field, which can be changed to a new_value. Must be present for EDIT `auth_action` only. `*` can be used if it doesn't matter what was the old value.

- `new_value` (string)

  New value that can be used to fill the field. `*` can be used if it doesn't matter what was the old value.

The `constraint_id` fields is where one can define the desired auth constraint for the action:

- `constraint` (dict)

  – `constraint_id` (string enum)

    Constraint Type. As of now, the following constraint types are supported:

    ```
     - 'ROLE': a constraint defining how many siganatures of a given role are␣
    ↪required
     - 'OR': logical disjunction for all constraints from `auth_constraints`
     - 'AND': logical conjunction for all constraints from `auth_constraints`
    ```

  – fields if `'constraint_id': 'OR'` or `'constraint_id': 'AND'`

    ∗ `auth_constraints` (list)

      A list of constraints. Any number of nested constraints is supported recursively

  – fields if `'constraint_id': 'ROLE'`:

    ∗ `role` (string enum)

      Who (what role) can perform the action Please have a look at *NYM* transaction description for a mapping between role codes and names.

    * `sig_count` (int):

      The number of signatures that is needed to do the action

    * `need_to_be_owner` (boolean):

      Flag to check if the user must be the owner of a transaction (Example: A steward must be the owner of the node to make changes to it). The notion of the `owner` is different for every auth rule. Please reference to *auth_rules.md* for details.

    * `off_ledger_signature` (boolean, optional, False by default):

      Whether signatures against keys not present on the ledger are accepted during verification of required number of valid signatures. An example when it can be set to `True` is creation of a new DID in a permissionless mode, that is when `identifer` is not present on the ledger and a newly created `verkey` is used for signature verification. Another example is signing by cryptonyms (where identifier is equal to verkey), but this is not supported yet. If the value of this field is False (default), and the number of required signatures is greater than zero, then the transaction author's DID (`identifier`) must be present on the ledger (corresponding NYM txn must exist).

    * `metadata` (dict; optional):

      Dictionary for additional parameters of the constraint. Can be used by plugins to add additional restrictions.

  – fields if `'constraint_id': 'FORBIDDEN'`:

    no fields

Example:

Let's consider an example of changing a value of a NODE transaction's `service` field from `[VALIDATOR]` to `[]` (demotion of a node). We are going to set an Auth Constraint, so that the action can be only be done by two TRUSTEE or one STEWARD who is the owner (the original creator) of this transaction.

```
{
   'txn':{
      'type':'120',
      'protocolVersion':2,
      'data':{
        'auth_type': '0',
        'auth_action': 'EDIT',
        'field' :'services',
        'old_value': [VALIDATOR],
        'new_value': []
        'constraint':{
              'constraint_id': 'OR',
              'auth_constraints': [{'constraint_id': 'ROLE',
                                    'role': '0',
                                    'sig_count': 2,
                                    'need_to_be_owner': False,
                                    'metadata': {}},

                                   {'constraint_id': 'ROLE',
                                    'role': '2',
                                    'sig_count': 1,
                                    'need_to_be_owner': True,
                                    'metadata': {}}
                                   ]
        },
      },
```

(continues on next page)

(continued from previous page)

```
      'metadata':{
         'reqId':252174114,
         'from':'M9BJDuS24bqbJNvBRsoGg3',
         'digest':'6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c',
         'payloadDigest':
↪'21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685',
      }
   },
   'txnMetadata':{
      'txnTime':1551785798,
      'seqNo':1
   },
   'reqSignature':{
      'type':'ED25519',
      'values':[
         {
            'value':
↪'4wpLLAtkT6SeiKEXPVsMcCirx9KvkeKKd11Q4VsMXmSv2tnJrRw1TQKFyov4m2BuPP4C5oCiZ6RUwS9w3EPdywnz
↪',
            'from':'M9BJDuS24bqbJNvBRsoGg3'
         }
      ]
   },
   'ver':'1'
}
```

1.6.5 AUTH_RULES

A command to set multiple AUTH_RULEs by one transaction. Transaction AUTH_RULES is not divided into a few AUTH_RULE transactions, and is written to the ledger with one transaction with the full set of rules that come in the request. Internally authentication rules are stored as a key-value dictionary: `{action} -> {auth_constraint}`.

The list of actions is static and can be found in *auth_rules.md*. There is a default Auth Constraint for every action (defined in *auth_rules.md*).

The `AUTH_RULES` command allows to change the Auth Constraints. So, it's not possible to register new actions by this command. But it's possible to override authentication constraints (values) for the given action.

Please note, that list elements of `GET_AUTH_RULE` output can be used as an input (with a required changes) for the field `rules` in `AUTH_RULES`.

- The `rules` field contains a list of auth rules. One rule has the following list of parameters which must match an auth rule from the *auth_rules.md*:
  - `auth_type` (string enum)

    The type of transaction to change the auth constraints to. (Example: "0", "1", ...). See transactions description to find the txn type enum value.
  - `auth_action` (enum: `ADD` or `EDIT`)

    Whether this is adding of a new transaction, or editing of an existing one.
  - `field` (string)

    Set the auth constraint of editing the given specific field. `*` can be used to specify that an auth rule is applied to all fields.

- `old_value` (string; optional)

  Old value of a field, which can be changed to a new_value. Must be present for EDIT `auth_action` only. `*` can be used if it doesn't matter what was the old value.

- `new_value` (string)

  New value that can be used to fill the field. `*` can be used if it doesn't matter what was the old value.

The `constraint_id` fields is where one can define the desired auth constraint for the action:

- `constraint` (dict)

  * `constraint_id` (string enum)

    Constraint Type. As of now, the following constraint types are supported:

    ```
     - 'ROLE': a constraint defining how many siganatures of a given role␣
    ↪are required
     - 'OR': logical disjunction for all constraints from `auth_constraints`
     - 'AND': logical conjunction for all constraints from `auth_constraints`
     - 'FORBIDDEN': a constraint for not allowed actions
    ```

  * fields if `'constraint_id': 'OR'` or `'constraint_id': 'AND'`

    · `auth_constraints` (list)

    A list of constraints. Any number of nested constraints is supported recursively

  * fields if `'constraint_id': 'ROLE'`:

    · `role` (string enum)

    Who (what role) can perform the action Please have a look at *NYM* transaction description for a mapping between role codes and names.

    · `sig_count` (int):

    The number of signatures that is needed to do the action

    · `need_to_be_owner` (boolean):

    Flag to check if the user must be the owner of a transaction (Example: A steward must be the owner of the node to make changes to it). The notion of the `owner` is different for every auth rule. Please reference to *auth_rules.md* for details.

    · `off_ledger_signature` (boolean, optional, False by default):

    Whether signatures against keys not present on the ledger are accepted during verification of required number of valid signatures. An example when it can be set to `True` is creation of a new DID in a permissionless mode, that is when `identifer` is not present on the ledger and a newly created `verkey` is used for signature verification. Another example is signing by cryptonyms (where identifier is equal to verkey), but this is not supported yet. If the value of this field is False (default), and the number of required signatures is greater than zero, then the transaction author's DID (`identifier`) must be present on the ledger (corresponding NYM txn must exist).

    · `metadata` (dict; optional):

    Dictionary for additional parameters of the constraint. Can be used by plugins to add additional restrictions.

  * fields if `'constraint_id': 'FORBIDDEN'`:

    no fields

Example:

```
{
   'txn':{
      'type':'120',
      'protocolVersion':2,
      'data':{
        rules: [
           {'auth_type': '0',
            'auth_action': 'EDIT',
            'field' :'services',
            'old_value': [VALIDATOR],
            'new_value': []
            'constraint':{
                  'constraint_id': 'OR',
                  'auth_constraints': [{'constraint_id': 'ROLE',
                                         'role': '0',
                                         'sig_count': 2,
                                         'need_to_be_owner': False,
                                         'metadata': {}},

                                        {'constraint_id': 'ROLE',
                                         'role': '2',
                                         'sig_count': 1,
                                         'need_to_be_owner': True,
                                         'metadata': {}}
                                        ]
            },
          },
          ...
        ]
      'metadata':{
         'reqId':252174114,
         'from':'M9BJDuS24bqbJNvBRsoGg3',
         'digest':'6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c',
         'payloadDigest':
↪'21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685',
      }
   },
   'txnMetadata':{
      'txnTime':1551785798,
      'seqNo':1
   },
   'reqSignature':{
      'type':'ED25519',
      'values':[
         {
            'value':
↪'4wpLLAtkT6SeiKEXPVsMcCirx9KvkeKKd11Q4VsMXmSv2tnJrRw1TQKFyov4m2BuPP4C5oCiZ6RUwS9w3EPdywnz
↪',
            'from':'M9BJDuS24bqbJNvBRsoGg3'
         }
      ]
   },
   'ver':'1'
}
```

1.6.6 TRANSACTION_AUTHOR_AGREEMENT

Setting (enabling/disabling) a transaction author agreement for the pool. If transaction author agreement is set, then all write requests to Domain ledger (transactions) must include additional metadata pointing to the latest transaction author agreement's digest which is signed by the transaction author.

If no transaction author agreement is set, or it's disabled, then no additional metadata is required.

Transaction author agreement can be disabled by setting an agreement with an empty text.

Each transaction author agreement has a unique version.

At least one *TRANSACTION_AUTHOR_AGREEMENT_AML* must be set on the ledger before submitting TRANSACTION_AUTHOR_AGREEMENT txn.

- `version` (string):

  Unique version of the transaction author agreement

- `text` (string):

  Transaction author agreement's text

Example:

```
{
    "ver":1,
    "txn": {
        "type":4,
        "protocolVersion":2,

        "data": {
            "ver":1,
            "version": "1.0",
            "text": "Please read carefully before writing anything to the ledger",
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "digest":"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c
↪",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.6.7 TRANSACTION_AUTHOR_AGREEMENT_AML

Setting a list of acceptance mechanisms for transaction author agreement.

Each write request for which a transaction author agreement needs to be accepted must point to a mechanism from the latest list on the ledger. The chosen mechanism is signed by the write request author (together with the transaction author agreement digest).

Each acceptance mechanisms list has a unique version.

- `version` (string):

  Unique version of the transaction author agreement acceptance mechanisms list

- `aml` (dict):

  Acceptance mechanisms list data in the form `<acceptance mechanism label>: <acceptance mechanism description>`

- `amlContext` (string, optional):

  A context information about Acceptance mechanisms list (may be URL to external resource).

Example:

```
{
    "ver":1,
    "txn": {
        "type":5,
        "protocolVersion":2,

        "data": {
            "ver":1,
            "version": "1.0",
            "aml": {
                "EULA": "Included in the EULA for the product being used",
                "Service Agreement": "Included in the agreement with the service␣
↪provider managing the transaction",
                "Click Agreement": "Agreed through the UI at the time of submission",
                "Session Agreement": "Agreed at wallet instantiation or login"
            },
            "amlContext": "http://aml-context-descr"
        },

        "metadata": {
            "reqId":1513945121191691,
            "from":"L5AD5g65TDQr1PPHHRoiGf",
            "digest":"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c
↪",
            "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
        },
    },
    "txnMetadata": {
        "txnTime":1513945121,
        "seqNo": 10,
    },
    "reqSignature": {
        "type": "ED25519",
        "values": [{
            "from": "L5AD5g65TDQr1PPHHRoiGf",
```

(continues on next page)

(continued from previous page)

```
            "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
        }]
    }
}
```

1.7 Actions

The actions are not written to the Ledger, so this is not a transaction, just a command.

1.7.1 POOL_RESTART

POOL_RESTART is the command to restart all nodes at the time specified in field "datetime"(sent by Trustee).

- `datetime` (string):

  Restart time in datetime frmat/ To restart as early as possible, send message without the "datetime" field or put in it value "0" or ""(empty string) or the past date on this place. The restart is performed immediately and there is no guarantee of receiving an answer with Reply.

- `action` (enum: `start` or `cancel`):

  Starts or cancels the Restart.

Example:

```
{
    "reqId": 98262,
    "type": "118",
    "identifier": "M9BJDuS24bqbJNvBRsoGg3",
    "datetime": "2018-03-29T15:38:34.464106+00:00",
    "action": "start"
}
```

CHAPTER 2

Requests

  - *GET_REVOC_REG_DEF*
  - *GET_REVOC_REG*
  - *GET_REVOC_REG_DELTA*
  - *GET_AUTH_RULE*
  - *GET_TRANSACTION_AUTHOR_AGREEMENT*
  - *GET_TRANSACTION_AUTHOR_AGREEMENT_AML*
  - *GET_TXN*
- *Action Requests*
  - *POOL_RESTART*
  - *VALIDATOR_INFO*

This doc is about supported client's Request (both write and read ones). If you are interested in transactions and their representation on the Ledger (that is internal one), then have a look at *transactions*.

indy-sdk expects the format as specified below.

See roles and permissions on the roles and who can create each type of transactions.

2.1 Common Request Structure

Each Request (both write and read) is a JSON with a number of common metadata fields.

```
{
    'operation': {
        'type': <request type>,
        <request-specific fields>
    },

    'identifier': <author DID>,
    `endorser`: <endorser DID>,
    'reqId': <req_id unique integer>,
    'protocolVersion': 2,
    'signature': <signature_value>,
    # 'signatures': {
    #       `did1`: <sig1>,
    #       `did2`: <sig2>,
    #   }

}
```

- `operation` (json):

  The request-specific operation json.

  - `type`: request type as one of the following values:
    - write requests (transactions):
      - NODE = "0"
      - NYM = "1"
      - TXN_AUTHOR_AGREEMENT = "4"

        · TXN_AUTHOR_AGREEMENT_AML = "5"
        · ATTRIB = "100"
        · SCHEMA = "101"
        · CLAIM_DEF = "102"
        · POOL_UPGRADE = "109"
        · NODE_UPGRADE = "110"
        · POOL_CONFIG = "111"
        · REVOC_REG_DEF = "113"
        · REVOC_REG_ENTRY = "114"
        · AUTH_RULE = "120"
        · AUTH_RULES = "122"

    ∗ read requests:

        · GET_TXN = "3"
        · GET_TXN_AUTHOR_AGREEMENT = "6"
        · GET_TXN_AUTHOR_AGREEMENT_AML = "7"
        · GET_ATTR = "104"
        · GET_NYM = "105"
        · GET_SCHEMA = "107"
        · GET_CLAIM_DEF = "108"
        · GET_REVOC_REG_DEF = "115"
        · GET_REVOC_REG = "116"
        · GET_REVOC_REG_DELTA = "117"
        · GET_AUTH_RULE = "121"

  – request-specific data

- `identifier` (base58-encoded string):

  Identifier of the transaction author as base58-encoded string for 16 or 32 bit DID value.

  For read requests this is read request submitter. It can be any DID (not necessary present on the ledger as a NYM txn)

  For write requests this is transaction author. It may differ from `endorser` field who submits the transaction on behalf of `identifier`. If `endorser` is absent, then the author (`identifier`) plays the role of endorser and submits request by his own. It also may differ from `dest` field for some of requests (for example NYM), where `dest` is a target identifier (for example, a newly created DID identifier).

  Example:

  – `identifier` is a DID of a transaction author who doesn't have write permissions; `endorser` is a DID of a user with Endorser role (that is with write permissions).

  – new NYM creation: `identifier` is a DID of an Endorser creating a new DID, and `dest` is a newly created DID.

- `reqId` (integer):

  Unique ID number of the request with transaction.

- `protocolVersion` (integer; optional):

  The version of client-to-node protocol. Each new version may introduce a new feature in Requests/Replies/Data. Since clients and different Nodes may be at different versions, we need this field to support backward compatibility between clients and nodes.

- `signature` (base58-encoded string; mutually exclusive with `signatures` field):

  Submitter's signature. Not required for read requests.

- `signatures` (map of base58-encoded string; mutually exclusive with `signature` field):

  Submitters' signature in multisig case. This is a map where client's identifiers are the keys and base58-encoded signature strings are the values. Not required for read requests.

Please find the format of each request-specific data for each type of request below.

2.2 Common Write Request Structure

Write requests to Domain and added-by-plugins ledgers may have additional Transaction Author Agreement acceptance fields:

```
{
    'operation': {
        'type': <request type>,
        <request-specific fields>
    },

    'identifier': <author DID>,
    'endorser': <endorser DID>,
    'reqId': <req_id unique integer>,
    'taaAcceptance': {
        'taaDigest': <digest hex string>,
        'mechanism': <mechaism string>,
        'time': <time integer>
     }
    'protocolVersion': 2,
    'signature': <signature_value>,
    # 'signatures': {
    #       `did1`: <sig1>,
    #       `did2`: <sig2>,
    #   }

}
```

Additional (optional) fields for write requests:

- `endorser` (base58-encoded string, optional): Identifier (DID) of an Endorser submitting a transaction on behalf of the original author (`identifier`) as base58-encoded string for 16 or 32 bit DID value. If `endorser` is absent, then the author (`identifier`) plays the role of endorser and submits request by his own. If `endorser` is present then the transaction must be multi-signed by the both author (`identifier`) and Endorser (`endorser`).

- `taaAcceptance` (dict, optional): If transaction author agreement is set/enabled, then every transaction (write request) from Domain and plugins-added ledgers must include acceptance of the latest transaction author agreement.
  - `taaDigest` (SHA256 hex digest string): SHA256 hex digest of the latest Transaction Author Agreement on the ledger. The digest is calculated from concatenation of *TRANSACTION_AUTHOR_AGREEMENT*'s `version` and `text`.
  - `mechanism` (string): a mechanism used to accept the signature; must be present in the latest list of transaction author agreement acceptane mechanisms on the ledger
  - `time` (integer as POSIX timestamp): transaction author agreement acceptance time

2.3 Reply Structure for Write Requests

Each Reply to write requests has a number of common metadata fields. Most of these fields are actually metadata fields of a transaction in the Ledger (see *transactions*).

```
{
    'op': 'REPLY',
    'result': {
        "ver": <...>,
        "txn": {
            "type": <...>,
            "protocolVersion": <...>,

            "data": {
                "ver": <...>,
                <txn-specific fields>
            },

            "metadata": {
                "reqId": <...>,
                "from": <...>,
                "endorser": <...>,
                "digest": <...>,
                "payloadDigest": <...>,
                "taaAcceptance": {
                    "taaDigest": <...>,
                    "mechanism": <...>,
                    "time": <...>
                 }
            },
        },
        "txnMetadata": {
            "txnTime": <...>,
            "seqNo": <...>,
            "txnId": <...>
        },
        "reqSignature": {
            "type": <...>,
            "values": [{
                "from": <...>,
                "value": <...>
            }]
        }
```

(continues on next page)

(continued from previous page)

```
        'rootHash': '5ecipNPSztrk6X77fYPdepzFRUvLdqBuSqv4M9Mcv2Vn',
        'auditPath': ['Cdsoz17SVqPodKpe6xmY2ZgJ9UcywFDZTRgWSAYM96iA',
↪'3phchUcMsnKFk2eZmcySAWm2T5rnzZdEypW7A5SKi1Qt'],
    }
}
```

- `ver` (string):

  Transaction version to be able to evolve content. The content of all sub-fields may depend on the version.

- `txn` (dict):

  Transaction-specific payload (data)

  – `type` (enum number as string):

    Supported transaction type:

    * NODE = "0"
    * NYM = "1"
    * TXN_AUTHOR_AGREEMENT = "4"
    * TXN_AUTHOR_AGREEMENT_AML = "5"
    * ATTRIB = "100"
    * SCHEMA = "101"
    * CLAIM_DEF = "102"
    * POOL_UPGRADE = "109"
    * NODE_UPGRADE = "110"
    * POOL_CONFIG = "111"
    * REVOC_REG_DEF = "113"
    * REVOC_REG_DEF = "114"
    * AUTH_RULE = "120"

  – `protocolVersion` (integer; optional):

    The version of client-to-node or node-to-node protocol. Each new version may introduce a new feature in Requests/Replies/Data. Since clients and different Nodes may be at different versions, we need this field to support backward compatibility between clients and nodes.

  – `data` (dict):

    Transaction-specific data fields (see next sections for each transaction description).

  – `metadata` (dict):

    Metadata as came from the Request.

    * `from` (base58-encoded string): Identifier (DID) of the transaction author as base58-encoded string for 16 or 32 bit DID value. It may differ from `endorser` field who submits the transaction on behalf of `identifier`. If `endorser` is absent, then the author (`identifier`) plays the role of endorser and submits request by his own. It also may differ from `dest` field for some of requests (for example NYM), where `dest` is a target identifier (for example, a newly created DID identifier).

      Example:

        · `identifier` is a DID of a transaction author who doesn't have write permissions; `endorser` is a DID of a user with Endorser role (that is with write permissions).

        · new NYM creation: `identifier` is a DID of an Endorser creating a new DID, and `dest` is a newly created DID.

    * `reqId` (integer): Unique ID number of the request with transaction.

    * `digest` (SHA256 hex digest string): SHA256 hash hex digest of all fields in the initial requests (including signatures)

    * `payloadDigest` (SHA256 hex digest string): SHA256 hash hex digest of the payload fields in the initial requests, that is all fields excluding signatures and plugins-added ones

    * `endorser` (base58-encoded string, optional): Identifier (DID) of an Endorser submitting a transaction on behalf of the original author (`identifier`) as base58-encoded string for 16 or 32 bit DID value. If `endorser` is absent, then the author (`identifier`) plays the role of endorser and submits request by his own. If `endorser` is present then the transaction must be multi-signed by the both author (`identifier`) and Endorser (`endorser`).

    * `taaAcceptance` (dict, optional): If transaction author agreement is set/enabled, then every transaction (write request) from Domain and plugins-added ledgers must include acceptance of the latest transaction author agreement.

```
 - `taaDigest` (SHA256 hex digest string): SHA256 hex digest of the␣
↪latest Transaction Author Agreement on the ledger

 - `mechanism` (string): a mechanism used to accept the signature; must␣
↪be present in the latest list of transaction author agreement acceptane␣
↪mechanisms on the ledger

 - `time` (integer as POSIX timestamp): transaction author agreement␣
↪acceptance time
```

- `txnMetadata` (dict):

  Metadata attached to the transaction.

  – `txnTime` (integer as POSIX timestamp): The time when transaction was written to the Ledger as POSIX timestamp.

  – `seqNo` (integer): A unique sequence number of the transaction on Ledger

  – `txnId` (string): Txn ID as State Trie key (address or descriptive data). It must be unique within the ledger.

- `reqSignature` (dict):

  Submitter's signature over request with transaction (`txn` field).

  – `type` (string enum):

    * ED25519: ed25519 signature

    * ED25519_MULTI: ed25519 signature in multisig case.

  – `values` (list):

    * `from` (base58-encoded string): Identifier (DID) of signer as base58-encoded string for 16 or 32 byte DID value.

    * `value` (base58-encoded string): signature value

- `rootHash` (base58-encoded hash string):

  base58-encoded ledger's merkle tree root hash

- `auditPath` (array of base58-encoded hash strings):

  ledger's merkle tree audit proof as array of base58-encoded hash strings (this is a cryptographic proof to verify that the new transaction has been appended to the ledger)

- transaction-specific fields as defined in *transactions* for each transaction type

2.4 Reply Structure for Read Requests

The structure below is not applicable for *GET_TXN*.

Each Reply to read requests has a number of common metadata fields and state-proof related fields. Some of these fields are actually metadata fields of a transaction in the Ledger (see *transactions*).

These common metadata values are added to result's JSON at the same level as real data.

TODO: consider distinguishing and separating real transaction data and metadata into different levels.

```
{
    'op': 'REPLY',
    'result': {
        'type': '105',
        'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514214863899317,

        'seqNo': 10,
        'txnTime': 1514214795,

        'state_proof': {
            'root_hash': '7Wdj3rrMCZ1R1M78H4xK5jxikmdUUGW2kbfJQ1HoEpK',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAg
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5yt
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514214795,
                    'ledger_id': 1,
                    'txn_root_hash': 'DqQ7G4fgDHBfdfVLrE6DCdYyyED1fY5oKw76aDeFsLVr',
                    'pool_state_root_hash':
↪'TfMhX3KDjrqq94Wj7BHV9sZrgivZyjbHJ3cGRG4h1Zj',
                    'state_root_hash': '7Wdj3rrMCZ1R1M78H4xK5jxikmdUUGW2kbfJQ1HoEpK'
                },
                'signature':
↪'RTyxbErBLcmTHBLj1rYCAEpMMkLnL65kchGni2tQczqzomYWZx9QQpLvnvNN5rD2nXkqaVW3USGak1vyAgvj2ecAKXQZXwcfo
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },

        'data': <transaction-specific data>,

        <request-specific data>
}
```

- `type` (enum number as string):

  Supported transaction types:

  - GET_TXN = “3”
  - GET_TXN_AUTHOR_AGREEMENT = “6”
  - GET_TXN_AUTHOR_AGREEMENT_AML = “7”
  - GET_ATTR = “104”
  - GET_NYM = “105”
  - GET_SCHEMA = “107”
  - GET_CLAIM_DEF = “108”
  - GET_REVOC_REG_DEF = “115”
  - GET_REVOC_REG = “116”
  - GET_REVOC_REG_DELTA = “117”
  - GET_AUTH_RULE = “121”

- `identifier` (base58-encoded string):

  read request submitter’s DID as was in read Request (may differ from the `identifier` in `data` which defines transaction author)

- `reqId` (integer):

  as was in read Request (may differ from the `reqId` in `data` which defines the request used to write the transaction to the Ledger)

- `seqNo` (integer):

  a unique sequence number of the transaction on Ledger

- `txnTime` (integer as POSIX timestamp):

  the time when transaction was written to the Ledger as POSIX timestamp

- `state_proof` (dict):

  State proof with BLS multi-signature of the State:

  - `root_hash` (base58-encoded string): state trie root hash for the ledger the returned transaction belongs to
  - `proof_nodes` (base64-encoded string): state proof for the returned transaction against the state trie with the specified `root_hash`
  - `multi_signature` (dict): BLS multi-signature against the specified state trie root hash
    - `value` (dict): the value the BLS multi-signature was created against
      - `timestamp` (integer as POSIX timestamp): last update of the state
      - `ledger_id` (integer enum): ID of the ledger the returned transaction belongs to (Pool=0; Domain=1; Config=2)
      - `txn_root_hash` (base58-encoded string): root hash of the ledger the returned transaction belongs to
      - `state_root_hash` (base58-encoded string): state trie root hash for the ledger the returned transaction belongs to
      - `pool_state_root_hash` (base58-encoded string): pool state trie root hash to get the state of the Pool at the moment the BLS multi-signature was created

    * `signature` (base58-encoded string): BLS multi-signature against the state trie with the specified `root_hash`
    * `participants` (array os strings): Aliases of Nodes participated in BLS multi-signature (the number of participated nodes is not less than n-f)

- `data` (json or dict):

  transaction-specific data (see *transactions* for each transaction type)

- request-specific fields as they appear in Read request

2.5 Write Requests

The format of each request-specific data for each type of request.

2.5.1 NYM

Creates a new NYM record for a specific user, endorser, steward or trustee. Note that only trustees and stewards can create new endorsers and trustee can be created only by other trusties (see roles).

The request can be used for creation of new DIDs, setting and rotation of verification key, setting and changing of roles.

- `dest` (base58-encoded string):

  Target DID as base58-encoded string for 16 or 32 byte DID value. It may differ from `identifier` metadata field, where `identifier` is the DID of the submitter. If they are equal (in permissionless case), then transaction must be signed by the newly created `verkey`.

  Example: `identifier` is a DID of a Endorser creating a new DID, and `dest` is a newly created DID.

- `role` (enum number as string; optional):

  Role of a user NYM record being created for. One of the following numbers

  – None (common USER)

  – “0” (TRUSTEE)

  – “2” (STEWARD)

  – “101” (ENDORSER)

  – “201” (NETWORK_MONITOR)

  A TRUSTEE can change any Nym’s role to None, this stopping it from making any writes (see roles).

- `verkey` (base58-encoded string, possibly starting with “~”; optional):

  Target verification key as base58-encoded string. It can start with “~”, which means that it’s abbreviated verkey and should be 16 bytes long when decoded, otherwise it’s a full verkey which should be 32 bytes long when decoded. If not set, then either the target identifier (`dest`) is 32-bit cryptonym CID (this is deprecated), or this is a user under guardianship (doesnt owns the identifier yet). Verkey can be changed to None by owner, it means that this user goes back under guardianship.

- `alias` (string; optional):

  NYM’s alias.

If there is no NYM transaction with the specified DID (`dest`), then it can be considered as creation of a new DID.

If there is a NYM transaction with the specified DID (`dest`), then this is update of existing DID. In this case we can specify only the values we would like to override. All unspecified values remain the same. So, if key rotation needs to be performed, the owner of the DID needs to send a NYM request with `dest` and `verkey` only. `role` and `alias` will stay the same.

Request Example:

```
{
    'operation': {
        'type': '1'
        'dest': 'GEzcdDLhCpGCYRHW82kjHd',
        'role': '101',
        'verkey': '~HmUWn928bnFT6Ephf65YXv'
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514213797569745,
    'protocolVersion': 2,
    'signature':
↪'49W5WP5jr7x1fZhtpAhHFbuUDqUYZ3AKht88gUjrz8TEJZr5MZUPjskpfBFdboLPZXKjbGjutoVascfKiMD5W7Ba
↪',
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver": 1,
        "txn": {
            "type":"1",
            "protocolVersion":2,

            "data": {
                "ver": 1,
                "dest":"GEzcdDLhCpGCYRHW82kjHd",
                "verkey":"~HmUWn928bnFT6Ephf65YXv",
                "role":101,
            },

            "metadata": {
                "reqId":1514213797569745,
                "from":"L5AD5g65TDQr1PPHHRoiGf",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
            "txnId": "N22KY2Dyvmuu2PyyqSFKue|01"
        },
        "reqSignature": {
            "type": "ED25519",
```

(continues on next page)

(continued from previous page)

```
            "values": [{
                "from": "L5AD5g65TDQr1PPHHRoiGf",
                "value":
↪"49W5WP5jr7x1fZhtpAhHFbuUDqUYZ3AKht88gUjrz8TEJZr5MZUPjskpfBFdboLPZXKjbGjutoVascfKiMD5W7Ba
↪"
            }]
        }

        'rootHash': '5ecipNPSztrk6X77fYPdepzFRUvLdqBuSqv4M9Mcv2Vn',
        'auditPath': ['Cdsoz17SVqPodKpe6xmY2ZgJ9UcywFDZTRgWSAYM96iA',
↪'3phchUcMsnKFk2eZmcySAWm2T5rnzZdEypW7A5SKi1Qt'],

        'dest': 'N22KY2Dyvmuu2PyyqSFKue',
        'role': '101',
        'verkey': '31V83xQnJDkZTSvm796X4MnzZFtUc96Tq6GJtuVkFQBE'
    }
}
```

2.5.2 ATTRIB

Adds attribute to a NYM record.

- `dest` (base58-encoded string):

  Target DID as base58-encoded string for 16 or 32 byte DID value. It differs from `identifier` metadata field, where `identifier` is the DID of the submitter.

  Example: `identifier` is a DID of a Endorser setting an attribute for a DID, and `dest` is the DID we set an attribute for.

- `raw` (json; mutually exclusive with `hash` and `enc`):

  Raw data is represented as json, where key is attribute name and value is attribute value.

- `hash` (sha256 hash string; mutually exclusive with `raw` and `enc`):

  Hash of attribute data.

- `enc` (string; mutually exclusive with `raw` and `hash`):

  Encrypted attribute data.

Request Example:

```
{
    'operation': {
        'type': '100'
        'dest': 'N22KY2Dyvmuu2PyyqSFKue',
        'raw': '{"name": "Alice"}'
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514213797569745,
    'protocolVersion': 2,
    'signature':
↪'49W5WP5jr7x1fZhtpAhHFbuUDqUYZ3AKht88gUjrz8TEJZr5MZUPjskpfBFdboLPZXKjbGjutoVascfKiMD5W7Ba
↪',
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver": 1,
        "txn": {
            "type":"100",
            "protocolVersion":2,

            "data": {
                "ver":1,
                "dest":"N22KY2Dyvmuu2PyyqSFKue",
                'raw': '{"name":"Alice"}'
            },

            "metadata": {
                "reqId":1514213797569745,
                "from":"L5AD5g65TDQr1PPHHRoiGf",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
            "txnId": "N22KY2Dyvmuu2PyyqSFKue|02"
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "L5AD5g65TDQr1PPHHRoiGf",
                "value":
↪"49W5WP5jr7x1fZhtpAhHFbuUDqUYZ3AKht88gUjrz8TEJZr5MZUPjskpfBFdboLPZXKjbGjutoVascfKiMD5W7Ba
↪"
            }]
        }

        'rootHash': '5ecipNPSztrk6X77fYPdepzFRUvLdqBuSqv4M9Mcv2Vn',
        'auditPath': ['Cdsoz17SVqPodKpe6xmY2ZgJ9UcywFDZTRgWSAYM96iA',
↪'3phchUcMsnKFk2eZmcySAWm2T5rnzZdEypW7A5SKi1Qt'],

    }
}
```

2.5.3 SCHEMA

Adds Claim's schema.

It's not possible to update existing Schema. So, if the Schema needs to be evolved, a new Schema with a new version or name needs to be created.

- `data` (dict):

  Dictionary with Schema's data:

  – `attr_names`: array of attribute name strings (125 attributes maximum)

   - `name`: Schema's name string
   - `version`: Schema's version string

Request Example:

```
{
    'operation': {
        'type': '101',
        'data': {
            'version': '1.0',
            'name': 'Degree',
            'attr_names': ['undergrad', 'last_name', 'first_name', 'birth_date',
↪'postgrad', 'expiry_date']
        },
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'endorser': 'D6HG5g65TDQr1PPHHRoiGf',
    'reqId': 1514280215504647,
    'protocolVersion': 2,
    'signature':
↪'5ZTp9g4SP6t73rH2s8zgmtqdXyTuSMWwkLvfV1FD6ddHCpwTY5SAsp8YmLWnTgDnPXfJue3vJBWjy89bSHvyMSdS
↪'
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver": 1,
        "txn": {
            "type":"101",
            "protocolVersion":2,

            "data": {
                "ver":1,
                "data": {
                    "name": "Degree",
                    "version": "1.0",
                    'attr_names': ['undergrad', 'last_name', 'first_name', 'birth_date
↪', 'postgrad', 'expiry_date']
                }
            },

            "metadata": {
                "reqId":1514280215504647,
                "from":"L5AD5g65TDQr1PPHHRoiGf",
                "endorser": "D6HG5g65TDQr1PPHHRoiGf",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
```

(continues on next page)

(continued from previous page)

```
            "txnId":"L5AD5g65TDQr1PPHHRoiGf1|Degree|1.0",
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "L5AD5g65TDQr1PPHHRoiGf",
                "value":
↪"5ZTp9g4SP6t73rH2s8zgmtqdXyTuSMWwkLvfV1FD6ddHCpwTY5SAsp8YmLWnTgDnPXfJue3vJBWjy89bSHvyMSdS
↪"
            }]
        }

        'rootHash': '5vasvo2NUAD7Gq8RVxJZg1s9F7cBpuem1VgHKaFP8oBm',
        'auditPath': ['Cdsoz17SVqPodKpe6xmY2ZgJ9UcywFDZTRgWSAYM96iA',
↪'66BCs5tG7qnfK6egnDsvcx2VSNH6z1Mfo9WmhLSExS6b'],

    }
}
```

2.5.4 CLAIM_DEF

Adds a claim definition (in particular, public key), that Issuer creates and publishes for a particular Claim Schema.

It's not possible to update `data` in existing Claim Def. So, if a Claim Def needs to be evolved (for example, a key needs to be rotated), then a new Claim Def needs to be created by a new Issuer DID (`identifier`).

- `data` (dict):

  Dictionary with Claim Definition's data:

  - `primary` (dict): primary claim public key
  - `revocation` (dict): revocation claim public key

- `ref` (string):

  Sequence number of a Schema transaction the claim definition is created for.

- `signature_type` (string):

  Type of the claim definition (that is claim signature). `CL` (Camenisch-Lysyanskaya) is the only supported type now.

- `tag` (string, optional):

  A unique tag to have multiple public keys for the same Schema and type issued by the same DID. A default tag `tag` will be used if not specified.

Request Example:

```
{
    'operation': {
        'type': '102',
        'signature_type': 'CL',
        'ref': 10,
        'tag': 'some_tag',
        'data': {
            'primary': ....,
            'revocation': ....
```

(continues on next page)

(continued from previous page)

```
        }
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'endorser': 'D6HG5g65TDQr1PPHHRoiGf',
    'reqId': 1514280215504647,
    'protocolVersion': 2,
    'signature':
↪'5ZTp9g4SP6t73rH2s8zgmtqdXyTuSMWwkLvfV1FD6ddHCpwTY5SAsp8YmLWnTgDnPXfJue3vJBWjy89bSHvyMSdS
↪'
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver": 1,
        "txn": {
            "type":"102",
            "protocolVersion":2,

            "data": {
                "ver":1,
                "signature_type":"CL",
                'ref': 10,
                'tag': 'some_tag',
                'data': {
                    'primary': ....,
                    'revocation': ....
                }
            },

            "metadata": {
                "reqId":1514280215504647,
                "from":"L5AD5g65TDQr1PPHHRoiGf",
                "endorser": "D6HG5g65TDQr1PPHHRoiGf",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
            "txnId":"HHAD5g65TDQr1PPHHRoiGf2L5AD5g65TDQr1PPHHRoiGf1|Degree1|CL|key1",
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "L5AD5g65TDQr1PPHHRoiGf",
                "value":
↪"5ZTp9g4SP6t73rH2s8zgmtqdXyTuSMWwkLvfV1FD6ddHCpwTY5SAsp8YmLWnTgDnPXfJue3vJBWjy89bSHvyMSdS
↪"
            }]
        },
```

(continues on next page)

(continued from previous page)

```
        'rootHash': '5vasvo2NUAD7Gq8RVxJZg1s9F7cBpuem1VgHKaFP8oBm',
        'auditPath': ['Cdsoz17SVqPodKpe6xmY2ZgJ9UcywFDZTRgWSAYM96iA',
↪'66BCs5tG7qnfK6egnDsvcx2VSNH6z1Mfo9WmhLSExS6b'],

    }
}
```

2.5.5 REVOC_REG_DEF

Adds a Revocation Registry Definition, that Issuer creates and publishes for a particular Claim Definition. It contains public keys, maximum number of credentials the registry may contain, reference to the Claim Def, plus some revocation registry specific data.

- `value` (dict):

  Dictionary with revocation registry definition's data:

  - `maxCredNum` (integer): a maximum number of credentials the Revocation Registry can handle
  - `tailsHash` (string): tails' file digest
  - `tailsLocation` (string): tails' file location (URL)
  - `issuanceType` (string enum): defines credentials revocation strategy. Can have the following values:
    - `ISSUANCE_BY_DEFAULT`: all credentials are assumed to be issued initially, so that Revocation Registry needs to be updated (REVOC_REG_ENTRY txn sent) only when revoking. Revocation Registry stores only revoked credentials indices in this case. Recommended to use if expected number of revocation actions is less than expected number of issuance actions.
    - `ISSUANCE_ON_DEMAND`: no credentials are issued initially, so that Revocation Registry needs to be updated (REVOC_REG_ENTRY txn sent) on every issuance and revocation. Revocation Registry stores only issued credentials indices in this case. Recommended to use if expected number of issuance actions is less than expected number of revocation actions.
  - `publicKeys` (dict): Revocation Registry's public key
- `id` (string): Revocation Registry Definition's unique identifier (a key from state trie is currently used)
- `credDefId` (string): The corresponding Credential Definition's unique identifier (a key from state trie is currently used)
- `revocDefType` (string enum): Revocation Type. `CL_ACCUM` (Camenisch-Lysyanskaya Accumulator) is the only supported type now.
- `tag` (string): A unique tag to have multiple Revocation Registry Definitions for the same Credential Definition and type issued by the same DID.

Request Example:

```
{
    'operation': {
        'type': '113',
        'id': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_
↪ACCUM:tag1',
        'credDefId': 'FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag'
        'revocDefType': 'CL_ACCUM',
        'tag': 'tag1',
```

(continues on next page)

(continued from previous page)

```
        'value': {
            'maxCredNum': 1000000,
            'tailsHash': '6619ad3cf7e02fc29931a5cdc7bb70ba4b9283bda3badae297',
            'tailsLocation': 'http://tails.location.com',
            'issuanceType': 'ISSUANCE_BY_DEFAULT',
            'publicKeys': {},
        },
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'endorser': 'D6HG5g65TDQr1PPHHRoiGf',
    'reqId': 1514280215504647,
    'protocolVersion': 2,
    'signature':
↪'5ZTp9g4SP6t73rH2s8zgmtqdXyTuSMWwkLvfV1FD6ddHCpwTY5SAsp8YmLWnTgDnPXfJue3vJBWjy89bSHvyMSdS
↪'
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver": 1,
        "txn": {
            "type":"113",
            "protocolVersion":2,

            "data": {
                "ver":1,
                'id': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1',
                'credDefId': 'FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag'
                'revocDefType': 'CL_ACCUM',
                'tag': 'tag1',
                'value': {
                    'maxCredNum': 1000000,
                    'tailsHash': '6619ad3cf7e02fc29931a5cdc7bb70ba4b9283bda3badae297',
                    'tailsLocation': 'http://tails.location.com',
                    'issuanceType': 'ISSUANCE_BY_DEFAULT',
                    'publicKeys': {},
                },
            },

            "metadata": {
                "reqId":1514280215504647,
                "from":"L5AD5g65TDQr1PPHHRoiGf",
                "endorser": "D6HG5g65TDQr1PPHHRoiGf",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
```

(continues on next page)

(continued from previous page)

```
            "txnId":"L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1",
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "L5AD5g65TDQr1PPHHRoiGf",
                "value":
↪"5ZTp9g4SP6t73rH2s8zgmtqdXyTuSMWwkLvfV1FD6ddHCpwTY5SAsp8YmLWnTgDnPXfJue3vJBWjy89bSHvyMSdS
↪"
            }]
        },

        'rootHash': '5vasvo2NUAD7Gq8RVxJZg1s9F7cBpuem1VgHKaFP8oBm',
        'auditPath': ['Cdsoz17SVqPodKpe6xmY2ZgJ9UcywFDZTRgWSAYM96iA',
↪'66BCs5tG7qnfK6egnDsvcx2VSNH6z1Mfo9WmhLSExS6b'],

    }
}
```

2.5.6 REVOC_REG_ENTRY

The RevocReg entry containing the new accumulator value and issued/revoked indices. This is just a delta of indices, not the whole list. So, it can be sent each time a new claim is issued/revoked.

- `value` (dict):

  Dictionary with revocation registry's data:

  - `accum` (string): the current accumulator value
  - `prevAccum` (string): the previous accumulator value; it's compared with the current value, and txn is rejected if they don't match; it's needed to avoid dirty writes and updates of accumulator.
  - `issued` (list of integers): an array of issued indices (may be absent/empty if the type is ISSUANCE_BY_DEFAULT); this is delta; will be accumulated in state.
  - `revoked` (list of integers): an array of revoked indices (delta; will be accumulated in state)
- `revocRegDefId` (string): The corresponding Revocation Registry Definition's unique identifier (a key from state trie is currently used)
- `revocDefType` (string enum): Revocation Type. `CL_ACCUM` (Camenisch-Lysyanskaya Accumulator) is the only supported type now.

Request Example:

```
{
    'operation': {
        'type': '114',
            'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1'
            'revocDefType': 'CL_ACCUM',
            'value': {
                'accum': 'accum_value',
                'prevAccum': 'prev_acuum_value',
                'issued': [],
                'revoked': [10, 36, 3478],
```

(continues on next page)

(continued from previous page)

```
            },
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'endorser': 'D6HG5g65TDQr1PPHHRoiGf',
    'reqId': 1514280215504647,
    'protocolVersion': 2,
    'signature':
↪'5ZTp9g4SP6t73rH2s8zgmtqdXyTuSMWwkLvfV1FD6ddHCpwTY5SAsp8YmLWnTgDnPXfJue3vJBWjy89bSHvyMSdS
↪'
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver": 1,
        "txn": {
            "type":"114",
            "protocolVersion":2,

            "data": {
                "ver":1,
                'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1'
                'revocDefType': 'CL_ACCUM',
                'value': {
                    'accum': 'accum_value',
                    'prevAccum': 'prev_acuum_value',
                    'issued': [],
                    'revoked': [10, 36, 3478],
                },
            },

            "metadata": {
                "reqId":1514280215504647,
                "from":"L5AD5g65TDQr1PPHHRoiGf",
                "endorser": "D6HG5g65TDQr1PPHHRoiGf",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
            "txnId":"5:L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1",
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "L5AD5g65TDQr1PPHHRoiGf",
                "value":
↪"5ZTp9g4SP6t73rH2s8zgmtqdXyTuSMWwkLvfV1FD6ddHCpwTY5SAsp8YmLWnTgDnPXfJue3vJBWjy89bSHvyMSdS
↪"
```

(continues on next page)

(continued from previous page)

```
            }]
        },

        'rootHash': '5vasvo2NUAD7Gq8RVxJZg1s9F7cBpuem1VgHKaFP8oBm',
        'auditPath': ['Cdsoz17SVqPodKpe6xmY2ZgJ9UcywFDZTRgWSAYM96iA',
↪'66BCs5tG7qnfK6egnDsvcx2VSNH6z1Mfo9WmhLSExS6b'],

    }
}
```

2.5.7 NODE

Adds a new node to the pool, or updates existing node in the pool.

- `data` (dict):

  Data associated with the Node:

  - `alias` (string): Node's alias
  - `blskey` (base58-encoded string; optional): BLS multi-signature key as base58-encoded string (it's needed for BLS signatures and state proofs support)
  - `client_ip` (string; optional): Node's client listener IP address, that is the IP clients use to connect to the node when sending read and write requests (ZMQ with TCP)
  - `client_port` (string; optional): Node's client listener port, that is the port clients use to connect to the node when sending read and write requests (ZMQ with TCP)
  - `node_ip` (string; optional): The IP address other Nodes use to communicate with this Node; no clients are allowed here (ZMQ with TCP)
  - `node_port` (string; optional): The port other Nodes use to communicate with this Node; no clients are allowed here (ZMQ with TCP)
  - `services` (array of strings; optional): the service of the Node. `VALIDATOR` is the only supported one now.

- `dest` (base58-encoded string):

  Target Node's verkey as base58-encoded string for 16 or 32 byte DID value. It differs from `identifier` metadata field, where `identifier` is the DID of the transaction submitter (Steward's DID).

  Example: `identifier` is a DID of a Steward creating a new Node, and `dest` is the verkey of this Node.

If there is no NODE transaction with the specified Node ID (`dest`), then it can be considered as creation of a new NODE.

If there is a NODE transaction with the specified Node ID (`dest`), then this is update of existing NODE. In this case we can specify only the values we would like to override. All unspecified values remain the same. So, if a Steward wants to rotate BLS key, then it's sufficient to send a NODE transaction with `dest` and a new `blskey` in `data`. There is no need to specify all other fields in `data`, and they will remain the same.

Request Example:

```
{
    'operation': {
        'type': '0'
            'data': {
```

(continues on next page)

(continued from previous page)

```
                        'alias': 'Node1',
                        'client_ip': '127.0.0.1',
                        'client_port': 7588,
                        'node_ip': '127.0.0.1',
                        'node_port': 7587,
                        'blskey': '00000000000000000000000000000000',
                        'services': ['VALIDATOR']}
                } ,
                'dest': '6HoV7DUEfNDiUP4ENnSC4yePja8w7JDQJ5uzVgyW4nL8'
    },

    'identifier': '21BPzYYrFzbuECcBV3M1FH',
    'reqId': 1514304094738044,
    'protocolVersion': 2,
    'signature':
↪'3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪',
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver": 1,
        "txn": {
            "type":0,
            "protocolVersion":2,

            "data": {
                "ver":1,
                'data': {
                    'alias': 'Node1',
                    'client_ip': '127.0.0.1',
                    'client_port': 7588,
                    'node_ip': '127.0.0.1',
                    'node_port': 7587,
                    'blskey': '00000000000000000000000000000000',
                    'services': ['VALIDATOR']}
                } ,
                'dest': '6HoV7DUEfNDiUP4ENnSC4yePja8w7JDQJ5uzVgyW4nL8'
            },

            "metadata": {
                "reqId":1514304094738044,
                "from":"21BPzYYrFzbuECcBV3M1FH",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
        },
        "reqSignature": {
```

(continues on next page)

(continued from previous page)

```
            "type": "ED25519",
            "values": [{
                "from": "21BPzYYrFzbuECcBV3M1FH",
                "value":
↪"3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪"
            }]
        }

        'rootHash': 'DvpkQ2aADvQawmrzvTTjF9eKQxjDkrCbQDszMRbgJ6zV',
        'auditPath': ['6GdvJfqTekMvzwi9wuEpfqMLzuN1T91kvgRBQLUzjkt6'],
    }
}
```

2.5.8 POOL_UPGRADE

Command to upgrade the Pool (sent by Trustee). It upgrades the specified Nodes (either all nodes in the Pool, or some specific ones).

- `name` (string):

  Human-readable name for the upgrade.

- `action` (enum: `start` or `cancel`):

  Starts or cancels the Upgrade.

- `version` (string):

  The version of indy-node package we perform upgrade to. Must be greater than existing one (or equal if `reinstall` flag is True).

- `schedule` (dict of node DIDs to timestamps):

  Schedule of when to perform upgrade on each node. This is a map where Node DIDs are keys, and upgrade time is a value (see example below). If `force` flag is False, then it's required that time difference between each Upgrade must be not less than 5 minutes (to give each Node enough time and not make the whole Pool go down during Upgrade).

- `sha256` (sha256 hash string):

  sha256 hash of the package

- `force` (boolean; optional):

  Whether we should apply transaction (schedule Upgrade) without waiting for consensus of this transaction. If false, then transaction is applied only after it's written to the ledger. Otherwise it's applied regardless of result of consensus, and there are no restrictions on the Upgrade `schedule` for each Node. So, we can Upgrade the whole Pool at the same time when it's set to True.False by default. Avoid setting to True without good reason.

- `reinstall` (boolean; optional):

  Whether it's allowed to re-install the same version. False by default.

- `timeout` (integer; optional):

  Limits upgrade time on each Node.

- `justification` (string; optional):

  Optional justification string for this particular Upgrade.

Request Example:

```
{
    'operation': {
        'type': '109'
        'name': `upgrade-13`,
        'action': `start`,
        'version': `1.3`,
        'schedule': {"4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2":"2017-12-
↪25T10:25:58.271857+00:00","AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3":"2017-12-
↪25T10:26:16.271857+00:00","DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2":"2017-12-
↪25T10:26:25.271857+00:00","JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ":"2017-12-
↪25T10:26:07.271857+00:00"},
        'sha256': `db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55`,
        'force': false,
        'reinstall': false,
        'timeout': 1
    },

    'identifier': '21BPzYYrFzbuECcBV3M1FH',
    'reqId': 1514304094738044,
    'protocolVersion': 2,
    'signature':
↪'3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪',
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver": 1,
        "txn": {
            "type":109,
            "protocolVersion":2,

            "data": {
                "ver":1,
                "name":"upgrade-13",
                "action":"start",
                "version":"1.3",
                "schedule":{"4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2":"2017-12-
↪25T10:25:58.271857+00:00","AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3":"2017-12-
↪25T10:26:16.271857+00:00","DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2":"2017-12-
↪25T10:26:25.271857+00:00","JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ":"2017-12-
↪25T10:26:07.271857+00:00"},
                "sha256":
↪"db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55",
                "force":false,
                "reinstall":false,
                "timeout":1,
                "justification":null,
            },

            "metadata": {
                "reqId":1514304094738044,
                "from":"21BPzYYrFzbuECcBV3M1FH",
```

(continues on next page)

(continued from previous page)

```
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "21BPzYYrFzbuECcBV3M1FH",
                "value":
↪"3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪"
            }]
        },

        'rootHash': 'DvpkQ2aADvQawmrzvTTjF9eKQxjDkrCbQDszMRbgJ6zV',
        'auditPath': ['6GdvJfqTekMvzwi9wuEpfqMLzuN1T91kvgRBQLUzjkt6'],
    }
}
```

2.5.9 POOL_CONFIG

Command to change Pool's configuration

- `writes` (boolean):

  Whether any write requests can be processed by the pool (if false, then pool goes to read-only state). True by default.

- `force` (boolean; optional):

  Whether we should apply transaction (for example, move pool to read-only state) without waiting for consensus of this transaction. If false, then transaction is applied only after it's written to the ledger. Otherwise it's applied regardless of result of consensus.False by default. Avoid setting to True without good reason.

Request Example:

```
{
    'operation': {
        'type': '111'
        'writes':false,
        'force':true
    },

    'identifier': '21BPzYYrFzbuECcBV3M1FH',
    'reqId': 1514304094738044,
    'protocolVersion': 2,
    'signature':
↪'3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪',
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver":1,
        "txn": {
            "type":111,
            "protocolVersion":2,

            "data": {
                "ver":1,
                "writes":false,
                "force":true,
            },

            "metadata": {
                "reqId":1514304094738044,
                "from":"21BPzYYrFzbuECcBV3M1FH",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685"
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "21BPzYYrFzbuECcBV3M1FH",
                "value":
↪"3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪"
            }]
        },

        'rootHash': 'DvpkQ2aADvQawmrzvTTjF9eKQxjDkrCbQDszMRbgJ6zV',
        'auditPath': ['6GdvJfqTekMvzwi9wuEpfqMLzuN1T91kvgRBQLUzjkt6'],
    }
}
```

2.5.10 AUTH_RULE

A command to change authentication rules. Internally authentication rules are stored as a key-value dictionary: `{action} -> {auth_constraint}`.

The list of actions is static and can be found in *auth_rules.md*. There is a default Auth Constraint for every action (defined in *auth_rules.md*).

The `AUTH_RULE` command allows to change the Auth Constraint. So, it's not possible to register new actions by this command. But it's possible to override authentication constraints (values) for the given action.

Please note, that list elements of `GET_AUTH_RULE` output can be used as an input (with a required changes) for `AUTH_RULE`.

If format of a transaction is incorrect, the client will receive NACK message for the request. A client will receive NACK for

- a request with incorrect format;
- a request with "ADD" action, but with "old_value";
- a request with "EDIT" action without "old_value";
- a request with a key that is not in the auth_rule.

The following input parameters must match an auth rule from the *auth_rules.md*:

- `auth_type` (string enum)

  The type of transaction to change the auth constraints to. (Example: "0", "1", ...). See transactions description to find the txn type enum value.

- `auth_action` (enum: `ADD` or `EDIT`)

  Whether this is addign of a new transaction, or editting of an existing one.

- `field` (string)

  Set the auth constraint of editing the given specific field. `*` can be used to specify that an auth rule is applied to all fields.

- `old_value` (string; optional)

  Old value of a field, which can be changed to a new_value. Must be present for EDIT `auth_action` only. `*` can be used if it doesn't matter what was the old value.

- `new_value` (string)

  New value that can be used to fill the field. `*` can be used if it doesn't matter what was the old value.

The `constraint_id` fields is where one can define the desired auth constraint for the action:

- `constraint` (dict)
  - `constraint_id` (string enum)

    Constraint Type. As of now, the following constraint types are supported:

    ```
      - 'ROLE': a constraint defining how many signatures of a given role are␣
    ↪required
      - 'OR': logical disjunction for all constraints from `auth_constraints`
      - 'AND': logical conjunction for all constraints from `auth_constraints`
      - 'FORBIDDEN': a constraint for not allowed actions
    ```

  - fields if `'constraint_id': 'OR'` or `'constraint_id': 'AND'`
    - `auth_constraints` (list)

      A list of constraints. Any number of nested constraints is supported recursively

  - fields if `'constraint_id': 'ROLE'`:
    - `role` (string enum)

      Who (what role) can perform the action Please have a look at *NYM* transaction description for a mapping between role codes and names.

    - `sig_count` (int):

      The number of signatures that is needed to do the action

    * `need_to_be_owner` (boolean):

      Flag to check if the user must be the owner of a transaction (Example: A steward must be the owner of the node to make changes to it). The notion of the `owner` is different for every auth rule. Please reference to *auth_rules.md* for details.

    * `off_ledger_signature` (boolean, optional, False by default):

      Whether signatures against keys not present on the ledger are accepted during verification of the required number of valid signatures. An example when it can be set to `True` is creation of a new DID in a permissionless mode, that is when `identifer` is not present on the ledger and a newly created `verkey` is used for signature verification. Another example is signing by cryptonyms (where identifier is equal to verkey), but this is not supported yet. If the value of this field is False (default), and the number of required signatures is greater than zero, then the transaction author's DID (`identifier`) must be present on the ledger (corresponding NYM txn must exist).

    * `metadata` (dict; optional):

      Dictionary for additional parameters of the constraint. Can be used by plugins to add additional restrictions.

  - fields if `'constraint_id': 'FORBIDDEN'`:

    no fields

Request Example:

Let's consider an example of changing a value of a NODE transaction's `service` field from `[VALIDATOR]` to `[]` (demotion of a node). We are going to set an Auth Constraint, so that the action can be only be done by two TRUSTEE or one STEWARD who is the owner (the original creator) of this transaction.

```
{
    'operation': {
        'type':'120',
        'auth_type': '0',
        'auth_action': 'EDIT',
        'field' :'services',
        'old_value': [VALIDATOR],
        'new_value': []
        'constraint':{
              'constraint_id': 'OR',
              'auth_constraints': [{'constraint_id': 'ROLE',
                                    'role': '0',
                                    'sig_count': 2,
                                    'need_to_be_owner': False,
                                    'metadata': {}},

                                   {'constraint_id': 'ROLE',
                                    'role': '2',
                                    'sig_count': 1,
                                    'need_to_be_owner': True,
                                    'metadata': {}}
                                   ]
        },
    },

    'identifier': '21BPzYYrFzbuECcBV3M1FH',
    'reqId': 1514304094738044,
    'protocolVersion': 2,
```

(continues on next page)

(continued from previous page)

```
    'signature':
↪'3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪'
}
```

Reply Example:

```
{      'op':'REPLY',
       'result':{
          'txnMetadata':{
             'seqNo':1,
             'txnTime':1551776783
          },
          'reqSignature':{
             'values':[
                {
                   'value':
↪'4j99V2BNRX1dn2QhnR8L9C3W9XQt1W3ScD1pyYaqD1NUnDVhbFGS3cw8dHRe5uVk8W7DoFtHb81ekMs9t9e76Fg
↪',
                   'from':'M9BJDuS24bqbJNvBRsoGg3'
                }
             ],
             'type':'ED25519'
          },
          'txn':{
             'data':{
                 'auth_type': '0',
                 'auth_action': 'EDIT',
                 'field' :'services',
                 'old_value': [VALIDATOR],
                 'new_value': []
                 'constraint':{
                           'constraint_id': 'OR',
                           'auth_constraints': [{'constraint_id': 'ROLE',
                                                 'role': '0',
                                                 'sig_count': 2,
                                                 'need_to_be_owner': False,
                                                 'metadata': {}},

                                                {'constraint_id': 'ROLE',
                                                 'role': '2',
                                                 'sig_count': 1,
                                                 'need_to_be_owner': True,
                                                 'metadata': {}}
                                                ]
                 },
             },
             'protocolVersion':2,
             'metadata':{
                'from':'M9BJDuS24bqbJNvBRsoGg3',
                'digest':
↪'ea13f0a310c7f4494d2828bccbc8ff0bd8b77d0c0bfb1ed9a84104bf55ad0436',
                'payloadDigest':
↪'21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685',
                'reqId':711182024
             },
             'type':'120'
```

(continues on next page)

(continued from previous page)

```
            },
            'ver':'1',
            'rootHash':'GJNfknLWDAb8R93cgAX3Bw6CYDo23HBhiwZnzb4fHtyi',
            'auditPath':['6GdvJfqTekMvzwi9wuEpfqMLzuN1T91kvgRBQLUzjkt6']
        }
    }
```

2.5.11 AUTH_RULES

A command to set multiple AUTH_RULEs by one transaction. Transaction AUTH_RULES is not divided into a few AUTH_RULE transactions, and is written to the ledger with one transaction with the full set of rules that come in the request. Internally authentication rules are stored as a key-value dictionary: `{action} -> {auth_constraint}`.

The list of actions is static and can be found in *auth_rules.md*. There is a default Auth Constraint for every action (defined in *auth_rules.md*).

The `AUTH_RULES` command allows to change the Auth Constraints. So, it's not possible to register new actions by this command. But it's possible to override authentication constraints (values) for the given action.

Please note, that list elements of `GET_AUTH_RULE` output can be used as an input (with a required changes) for the field `rules` in `AUTH_RULES`.

If one rule is incorrect, the client will receive NACK message for the request with all its rules. A client will receive NACK for

- a request with incorrect format;
- a request with a rule with "ADD" action, but with "old_value";
- a request with a rule with "EDIT" action without "old_value";
- a request with a rule with a key that is not in the auth_rule.
- The `rules` field contains a list of auth rules. One rule has the following list of parameters which must match an auth rule from the *auth_rules.md*:
  - `auth_type` (string enum)

    The type of transaction to change the auth constraints to. (Example: "0", "1", . . .). See transactions description to find the txn type enum value.
  - `auth_action` (enum: `ADD` or `EDIT`)

    Whether this is adding of a new transaction, or editing of an existing one.
  - `field` (string)

    Set the auth constraint of editing the given specific field. `*` can be used to specify that an auth rule is applied to all fields.
  - `old_value` (string; optional)

    Old value of a field, which can be changed to a new_value. Must be present for EDIT `auth_action` only. `*` can be used if it doesn't matter what was the old value.
  - `new_value` (string)

    New value that can be used to fill the field. `*` can be used if it doesn't matter what was the old value.

  The `constraint_id` fields is where one can define the desired auth constraint for the action:
  - `constraint` (dict)

* `constraint_id` (string enum)

  Constraint Type. As of now, the following constraint types are supported:

  ```
  - 'ROLE': a constraint defining how many signatures of a given role are␣
  ↪required
  - 'OR': logical disjunction for all constraints from `auth_constraints`
  - 'AND': logical conjunction for all constraints from `auth_constraints`
  - 'FORBIDDEN': a constraint for not allowed actions
  ```

* fields if `'constraint_id': 'OR'` or `'constraint_id': 'AND'`

  · `auth_constraints` (list)

  A list of constraints. Any number of nested constraints is supported recursively

* fields if `'constraint_id': 'ROLE'`:

  · `role` (string enum)

  Who (what role) can perform the action Please have a look at *NYM* transaction description for a mapping between role codes and names.

  · `sig_count` (int):

  The number of signatures that is needed to do the action

  · `need_to_be_owner` (boolean):

  Flag to check if the user must be the owner of a transaction (Example: A steward must be the owner of the node to make changes to it). The notion of the `owner` is different for every auth rule. Please reference to *auth_rules.md* for details.

  · `off_ledger_signature` (boolean, optional, False by default):

  Whether signatures against keys not present on the ledger are accepted during verification of required number of valid signatures. An example when it can be set to `True` is creation of a new DID in a permissionless mode, that is when `identifer` is not present on the ledger and a newly created `verkey` is used for signature verification. Another example is signing by cryptonyms (where identifier is equal to verkey), but this is not supported yet. If the value of this field is False (default), and the number of required signatures is greater than zero, then the transaction author's DID (`identifier`) must be present on the ledger (corresponding NYM txn must exist).

  · `metadata` (dict; optional):

  Dictionary for additional parameters of the constraint. Can be used by plugins to add additional restrictions.

* fields if `'constraint_id': 'FORBIDDEN'`:

  no fields

Request Example:

```
{
    'operation': {
           'type':'122',
           'rules': [
                {'constraint':{
                     'constraint_id': 'OR',
                     'auth_constraints': [{'constraint_id': 'ROLE',
                                           'role': '0',
                                           'sig_count': 1,
```

(continues on next page)

(continued from previous page)

```
                                              'need_to_be_owner': False,
                                              'metadata': {}},

                                              {'constraint_id': 'ROLE',
                                               'role': '2',
                                               'sig_count': 1,
                                               'need_to_be_owner': True,
                                               'metadata': {}}
                                              ]
                          },
                        'field' :'services',
                        'auth_type': '0',
                        'auth_action': 'EDIT',
                        'old_value': [VALIDATOR],
                        'new_value': []
                     },
                     ...
                ]
    },

    'identifier': '21BPzYYrFzbuECcBV3M1FH',
    'reqId': 1514304094738044,
    'protocolVersion': 1,
    'signature':
↪'3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪'
}
```

Reply Example:

```
{      'op':'REPLY',
       'result':{
          'txnMetadata':{
             'seqNo':1,
             'txnTime':1551776783
          },
          'reqSignature':{
             'values':[
                {
                   'value':
↪'4j99V2BNRX1dn2QhnR8L9C3W9XQt1W3ScD1pyYaqD1NUnDVhbFGS3cw8dHRe5uVk8W7DoFtHb81ekMs9t9e76Fg
↪',
                   'from':'M9BJDuS24bqbJNvBRsoGg3'
                }
             ],
             'type':'ED25519'
          },
          'txn':{
             'type':'122',
             'data':{
                'rules': [
                     {'constraint':{
                          'constraint_id': 'OR',
                          'auth_constraints': [{'constraint_id': 'ROLE',
                                                'role': '0',
                                                'sig_count': 1,
                                                'need_to_be_owner': False,
```

(continues on next page)

(continued from previous page)

```
                                                    'metadata': {}},

                                                    {'constraint_id': 'ROLE',
                                                     'role': '2',
                                                     'sig_count': 1,
                                                     'need_to_be_owner': True,
                                                     'metadata': {}}
                                                    ]
                              },
                            'field' :'services',
                            'auth_type': '0',
                            'auth_action': 'EDIT',
                            'old_value': [VALIDATOR],
                            'new_value': []
                           },
                           ...
                       ]
                   }
                   'protocolVersion':2,
                   'metadata':{
                      'from':'M9BJDuS24bqbJNvBRsoGg3',
                      'digest':
↪'ea13f0a310c7f4494d2828bccbc8ff0bd8b77d0c0bfb1ed9a84104bf55ad0436',
                      'reqId':711182024
                   }
                },
                'ver':'1',
                'rootHash':'GJNfknLWDAb8R93cgAX3Bw6CYDo23HBhiwZnzb4fHtyi',
                'auditPath':[

                ]
            }
        }
```

2.5.12 TRANSACTION_AUTHOR_AGREEMENT

Setting (enabling/disabling) a transaction author agreement for the pool. If transaction author agreement is set, then all write requests to Domain ledger (transactions) must include additional metadata pointing to the latest transaction author agreement's digest which is signed by the transaction author.

If no transaction author agreement is set, or it's disabled, then no additional metadata is required.

Transaction author agreement can be disabled by setting an agreement with an empty text.

Each transaction author agreement has a unique version.

At least one *TRANSACTION_AUTHOR_AGREEMENT_AML* must be set on the ledger before submitting TRANSACTION_AUTHOR_AGREEMENT txn.

- `version` (string):

  Unique version of the transaction author agreement

- `text` (string):

  Transaction author agreement's text

Request Example:

```
{
    'operation': {
        'type': '4'
        'version': '1.0',
        'text': 'Please read carefully before writing anything to the ledger',
    },

    'identifier': '21BPzYYrFzbuECcBV3M1FH',
    'reqId': 1514304094738044,
    'protocolVersion': 2,
    'signature':
↪'3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪',
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver":1,
        "txn": {
            "type":4,
            "protocolVersion":2,

            "data": {
                "ver":1,
                'version': '1.0',
                'text': 'Please read carefully before writing anything to the ledger',
            },

            "metadata": {
                "reqId":1514304094738044,
                "from":"21BPzYYrFzbuECcBV3M1FH",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "21BPzYYrFzbuECcBV3M1FH",
                "value":
↪"3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪"
            }]
        },

        'rootHash': 'DvpkQ2aADvQawmrzvTTjF9eKQxjDkrCbQDszMRbgJ6zV',
        'auditPath': ['6GdvJfqTekMvzwi9wuEpfqMLzuN1T91kvgRBQLUzjkt6'],
    }
}
```

2.5.13 TRANSACTION_AUTHOR_AGREEMENT_AML

Setting a list of acceptance mechanisms for transaction author agreement.

Each write request for which a transaction author agreement needs to be accepted must point to a mechanism from the latest list on the ledger. The chosen mechanism is signed by the write request author (together with the transaction author agreement digest).

Each acceptance mechanisms list has a unique version.

- `version` (string):

  Unique version of the transaction author agreement acceptance mechanisms list

- `aml` (dict):

  Acceptance mechanisms list data in the form `<acceptance mechanism label>: <acceptance mechanism description>`

- `amlContext` (string, optional):

  A context information about Acceptance mechanisms list (may be URL to external resource).

Request Example:

```
{
    'operation': {
        'type': '5'
        "version": "1.0",
        "aml": {
            "EULA": "Included in the EULA for the product being used",
            "Service Agreement": "Included in the agreement with the service provider␣
↪managing the transaction",
            "Click Agreement": "Agreed through the UI at the time of submission",
            "Session Agreement": "Agreed at wallet instantiation or login"
        },
        "amlContext": "http://aml-context-descr"
    },

    'identifier': '21BPzYYrFzbuECcBV3M1FH',
    'reqId': 1514304094738044,
    'protocolVersion': 2,
    'signature':
↪'3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪',
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        "ver":1,
        "txn": {
            "type":5,
            "protocolVersion":2,

            "data": {
                "ver":1,
                "version": "1.0",
                "aml": {
```

(continues on next page)

(continued from previous page)

```
                    "EULA": "Included in the EULA for the product being used",
                    "Service Agreement": "Included in the agreement with the service␣
↪provider managing the transaction",
                    "Click Agreement": "Agreed through the UI at the time of␣
↪submission",
                    "Session Agreement": "Agreed at wallet instantiation or login"
                },
                "amlContext": "http://aml-context-descr"
            },

            "metadata": {
                "reqId":1514304094738044,
                "from":"21BPzYYrFzbuECcBV3M1FH",
                "digest":
↪"6cee82226c6e276c983f46d03e3b3d10436d90b67bf33dc67ce9901b44dbc97c",
                "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
            },
        },
        "txnMetadata": {
            "txnTime":1513945121,
            "seqNo": 10,
        },
        "reqSignature": {
            "type": "ED25519",
            "values": [{
                "from": "21BPzYYrFzbuECcBV3M1FH",
                "value":
↪"3YVzDtSxxnowVwAXZmxCG2fz1A38j1qLrwKmGEG653GZw7KJRBX57Stc1oxQZqqu9mCqFLa7aBzt4MKXk4MeunVj
↪"
            }]
        },

        'rootHash': 'DvpkQ2aADvQawmrzvTTjF9eKQxjDkrCbQDszMRbgJ6zV',
        'auditPath': ['6GdvJfqTekMvzwi9wuEpfqMLzuN1T91kvgRBQLUzjkt6'],
    }
}
```

2.6 Read Requests

2.6.1 GET_NYM

Gets information about a DID (NYM).

- `dest` (base58-encoded string):

  Target DID as base58-encoded string for 16 or 32 byte DID value. It differs from `identifier` metadata field, where `identifier` is the DID of the submitter.

  Example: `identifier` is a DID of the read request sender, and `dest` is the requested DID.

Request Example:

```
{
    'operation': {
```

(continues on next page)

(continued from previous page)

```
        'type': '105'
        'dest': '2VkbBskPNNyWrLrZq7DBhk'
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        'type': '105',
        'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'seqNo': 10,
        'txnTime': 1514214795,

        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7-
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 1,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtFl
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },

        'data': '{"dest":"2VkbBskPNNyWrLrZq7DBhk","identifier":"L5AD5g65TDQr1PPHHRoiGf
↪","role":null,"seqNo":10,"txnTime":1514308168,"verkey":"~6hAzy6ubo3qutnnw5A12RF"}',

        'dest': '2VkbBskPNNyWrLrZq7DBhk'
    }
}
```

2.6.2 GET_ATTRIB

Gets information about an Attribute for the specified DID.

NOTE: `GET_ATTRIB` for `hash` and `enc` attributes is something like the "proof of existence", i.e. reply data contains requested value only.

- `dest` (base58-encoded string):

  Target DID as base58-encoded string for 16 or 32 byte DID value. It differs from `identifier` metadata field, where `identifier` is the DID of the submitter.

  Example: `identifier` is a DID of read request sender, and `dest` is the DID we get an attribute for.

- `raw` (string; mutually exclusive with `hash` and `enc`):

  Requested attribute name.

- `hash` (sha256 hash string; mutually exclusive with `raw` and `enc`):

  Requested attribute hash.

- `enc` (string; mutually exclusive with `raw` and `hash`):

  Encrypted attribute.

Request Example:

```
{
    'operation': {
        'type': '104'
        'dest': 'AH4RRiPR78DUrCWatnCW2w',
        'raw': 'dateOfBirth'
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        'type': '104',
        'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'seqNo': 10,
        'txnTime': 1514214795,

        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7-
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 1,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
```

(continues on next page)

(continued from previous page)

```
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtF
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },

        'data': '{"dateOfBirth":{"dayOfMonth":23,"month":5,"year":1984}}',

        'dest': 'AH4RRiPR78DUrCWatnCW2w',
        'raw': 'dateOfBirth'
    }
}
```

2.6.3 GET_SCHEMA

Gets Claim's Schema.

- `dest` (base58-encoded string):

  Schema Issuer's DID as base58-encoded string for 16 or 32 byte DID value. It differs from `identifier` metadata field, where `identifier` is the DID of the submitter.

  Example: `identifier` is a DID of the read request sender, and `dest` is the DID of the Schema's Issuer.

- `data` (dict):
  - `name` (string): Schema's name string
  - `version` (string): Schema's version string

Request Example:

```
{
    'operation': {
        'type': '107'
        'dest': '2VkbBskPNNyWrLrZq7DBhk',
        'data': {
            'name': 'Degree',
             'version': '1.0'
        },
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        'type': '107',
        'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,
```

(continues on next page)

(continued from previous page)

```
        'seqNo': 10,
        'txnTime': 1514214795,

        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7-
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 1,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtFI
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },

        'data': {
            'name': 'Degree',
            'version': '1.0',
            'attr_names': ['attrib1', 'attrib2', 'attrib3']
        },

        'dest': '2VkbBskPNNyWrLrZq7DBhk'
    }
}
```

2.6.4 GET_CLAIM_DEF

Gets Claim Definition.

- `origin` (base58-encoded string):

  Claim Definition Issuer's DID as base58-encoded string for 16 or 32 byte DID value.

- `ref` (string):

  Sequence number of a Schema transaction the claim definition is created for.

- `signature_type` (string):

  Type of the claim definition (that is claim signature). `CL` (Camenisch-Lysyanskaya) is the only supported type now.

- `tag` (string, optional):

  A unique tag to have multiple public keys for the same Schema and type issued by the same DID. A default tag `tag` will be used if not specified.

Request Example:

```
{
    'operation': {
        'type': '108'
        'signature_type': 'CL',
        'origin': '2VkbBskPNNyWrLrZq7DBhk',
        'ref': 10,
        'tag': 'some_tag',
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        'type': '108',
        'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'seqNo': 10,
        'txnTime': 1514214795,

        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7-
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 1,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtFI
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },

        'data': {
            'primary': ...,
            'revocation': ...
        },

        'signature_type': 'CL',
        'origin': '2VkbBskPNNyWrLrZq7DBhk',
```

(continues on next page)

(continued from previous page)

```
        'ref': 10,
        'tag': 'some_tag'
    }
}
```

2.6.5 GET_REVOC_REG_DEF

Gets a Revocation Registry Definition, that Issuer creates and publishes for a particular Claim Definition.

- `id` (string): Revocation Registry Definition's unique identifier (a key from state trie is currently used)

Request Example:

```
{
    'operation': {
        'type': '115'
        'id': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_
↪ACCUM:tag1',
    },

    'identifier': 'T6AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        'type': '115',
        'identifier': 'T6AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'id': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_
↪ACCUM:tag1',

        'seqNo': 10,
        'txnTime': 1514214795,

        'data': {
            'id': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_
↪ACCUM:tag1',
            'credDefId': 'FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag'
            'revocDefType': 'CL_ACCUM',
            'tag': 'tag1',
            'value': {
                'maxCredNum': 1000000,
                'tailsHash': '6619ad3cf7e02fc29931a5cdc7bb70ba4b9283bda3badae297',
                'tailsLocation': 'http://tails.location.com',
                'issuanceType': 'ISSUANCE_BY_DEFAULT',
                'publicKeys': {},
            },
        },
```

(continues on next page)

(continued from previous page)

```
        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7-
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 1,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtFI
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },


    }
}
```

2.6.6 GET_REVOC_REG

Gets a Revocation Registry Accumulator.

- `revocRegDefId` (string): The corresponding Revocation Registry Definition's unique identifier (a key from state trie is currently used)
- `timestamp` (integer as POSIX timestamp):

  The time (from the ledger point of view) for which we want to get the accumulator value.

Request Example:

```
{
    'operation': {
        'type': '116'
        'revocRegDefId': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1',
        'timestamp': 1514214800
    },

    'identifier': 'T6AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        'type': '116',
        'identifier': 'T6AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'revocRegDefId': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1',
        'timestamp': 1514214800

        'seqNo': 10,
        'txnTime': 1514214795,

        'data': {
            'id': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_
↪ACCUM:tag1',
            'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1'
            'revocDefType': 'CL_ACCUM',
            'value': {
                'accum': 'accum_value',
            },
        },


        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7-
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 1,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtFI
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },


    }
}
```

2.6.7 GET_REVOC_REG_DELTA

Gets a Revocation Registry Delta (accum values, and delta of issues/revoked indices) for the given time interval (`from` and `to`).

If from is not set, then the whole registry (accum and current issues/revoked indices) is returned for the given time (`to`)

- `revocRegDefId` (string): The corresponding Revocation Registry Definition's unique identifier (a key from state trie is currently used)
- `from` (integer as POSIX timestamp, optional):

  The time (from the ledger point of view) we want to return accum and indices after, that is the left bound of the delta interval. Can be absent, which means that all indices and accum needs to be returned till `to`.
- `to` (integer as POSIX timestamp):

  The time (from the ledger point of view) we want to return accum and indices before, that is the right bound of the delta interval.

Please note, that if `from` is set, then addition state proof for `accum_from` value is returned in `stateProofFrom`, while the common state proof is for `accum_to` Both state proofs are returned just for accumulator values. The client needs to check that the returned delta is also correct by calculating `accum_to` from the given `accum_from` and delta indices, and making sure that the calculated `accum_to` is equal to the returned one.

If `from` is not set, then there is just one state proof (as usual) for both `accum` value and the whole indices lists.

Request Example when both `from` and `to` present:

```
{
    'operation': {
        'type': '117'
        'revocRegDefId': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1',
        'from': 1514214100
        'to': 1514214900
    },

    'identifier': 'T6AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example when both `from` and `to` present:

```
{
    'op': 'REPLY',
    'result': {
        'type': '117',
        'identifier': 'T6AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'revocRegDefId': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1',
        'from': 1514214100
        'to': 1514214900

        'seqNo': 18,
        'txnTime': 1514214795,
```

(continues on next page)

(continued from previous page)

```
        'data': {
            'revocDefType': 'CL_ACCUM',
            'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1',
            'value': {
               'accum_to': {
                   'revocDefType': 'CL_ACCUM',
                   'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1',
                   'txnTime': 1514214795,
                   'seqNo': 18,
                   'value': {
                      'accum': '9a512a7624'
                   }
                },
                'revoked': [10, 11],
                'issued': [1, 2, 3],
                'accum_from': {
                    'revocDefType': 'CL_ACCUM',
                    'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1',
                    'txnTime': 1514214105,
                    'seqNo': 16,
                    'value': {
                       'accum': 'be080bd74b'
                    }
                 }
            },
            'stateProofFrom': {
                'multi_signature': {
                     'participants': ['Delta', 'Gamma', 'Alpha'],
                     'signature':
↪'QpP4oVm2MLQ7SzLVZknuFjneXfqYj6UStn3oQtCdSiKiYuS4n1kxRphKRDMwmS7LGeXgUmy3C8GtcVM5X9SN9qLr2MBApjpPtI
↪',
                     'value': {
                         'state_root_hash':
↪'2sfnQcEKkjw78KYnGJyk5Gw9gtwESvX6NdFFPEiQYQsz',
                         'ledger_id': 1,
                         'pool_state_root_hash':
↪'JDt3NNrZenx3x41oxsvhWeuSFFerdyqEvQUWyGdHX7gx',
                         'timestamp': 1514214105,
                         'txn_root_hash':
↪'FCkntnPqfaGx4fCX5tTdWeLr1mXdFuZnTNuEehiet32z'
                     }
                },
                'root_hash': '2sfnQcEKkjw78KYnGJyk5Gw9gtwESvX6NdFFPEiQYQsz',
                'proof_nodes':
↪'+QLB+QE3uFEgOk1TaktUV2tQTHRZb1BFYVRGMVRVRGI6NDpNU2pLVFdrUEx0WW9QRWFURjFUVURiOjM6Q0w6MTM6c29tZV90YW
↪KAFG4RQqB9dAGRfTgly2XjXvPCeVr7vBn6FSkN7sH4CAoBGxQDip9XfEC/
↪CkgimSkkhCeMm9XnkxxwWiMJwzuhAjgKAmh0g8FUI60e7NBwTu7ukdfz6kaON6u9U87kTeTlPcXICgsk2X2G6MlVhEqMEzthWA
↪q4od/mRjFpLdXKR3YG2o/hAygRIVVdoVD0dpqktsN8kSc03UhYiI76nxdCejX+CV4OX6A'
            }
        },


        'state_proof': {
```

(continues on next page)

(continued from previous page)

```
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAg
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5yt
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 1,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtF
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },


    }
}
```

Request Example when there is only `to` *present*:

```
{
    'operation': {
        'type': '117'
        'revocRegDefId': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1',
        'to': 1514214900
    },

    'identifier': 'T6AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example when there is only `to` *present*:

```
{
    'op': 'REPLY',
    'result': {
        'type': '117',
        'identifier': 'T6AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'revocRegDefId': 'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_
↪tag:CL_ACCUM:tag1',
        'to': 1514214900

        'seqNo': 18,
        'txnTime': 1514214795,
```

(continues on next page)

(continued from previous page)

```
        'data': {
            'revocDefType': 'CL_ACCUM',
            'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1',
            'value': {
               'accum_to': {
                   'revocDefType': 'CL_ACCUM',
                   'revocRegDefId':
↪'L5AD5g65TDQr1PPHHRoiGf:3:FC4aWomrA13YyvYC1Mxw7:3:CL:14:some_tag:CL_ACCUM:tag1',
                   'txnTime': 1514214795,
                   'seqNo': 18,
                   'value': {
                      'accum': '9a512a7624'
                   }
                },
                'issued': [],
                'revoked': [1, 2, 3, 4, 5]
        },


        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7+
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 1,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtF
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },


    }
}
```

2.6.8 GET_AUTH_RULE

A request to get an auth constraint for an authentication rule or a full list of rules from Ledger. The constraint format is described in *AUTH_RULE transaction*.

The set of Auth Rules is static and can be found in *auth_rules.md*. This is the constraint part that may be changed and edited.

Two options are possible in a request builder:

- If the request has a full list of parameters (probably without `old_value` as it's not required for ADD actions), then the reply will contain one constraint for this key.
- If the request does not contain fields other than txn_type, the response will contain a full list of authentication rules.

A reply is a list of Auth Rules with constraints. This will be a one-element list in case of GET_AUTH_RULE with params, that is GET_AUTH_RULE for specific action.

Each output list element is equal to the input of *AUTH_RULE*, so list elements of `GET_AUTH_RULE` output can be used as an input (with a required changes) for `AUTH_RULE`.

- `auth_action` (enum: `ADD` or `EDIT`; optional):

  Action type: add a new entity or edit an existing one.

- `auth_type` (string; optional):

  The type of transaction to change rights for. (Example: "0", "1", . . .)

- `field` (string; optional):

  Change the rights for editing (adding) a value of the given transaction field. `*` can be used as `any field`.

- `old_value` (string; optional):

  Old value of a field, which can be changed to a new_value. Makes sense for EDIT actions only.

- `new_value` (string; optional):

  New value that can be used to fill the field.

Request Example (for getting one rule):

```
  {
      'reqId':572495653,
      'signature':
↪'366f89ehxLuxPySGcHppxbURWRcmXVdkHeHrjtPKNYSRKnvaxzUXF8CEUWy9KU251u5bmnRL3TKvQiZgjwouTJYH
↪',
      'identifier':'M9BJDuS24bqbJNvBRsoGg3',
      'operation':{
            'auth_type': '0',
            'auth_action': 'EDIT',
            'field' :'services',
            'old_value': [VALIDATOR],
            'new_value': []
      },
      'protocolVersion':2
   }
```

Reply Example (for getting one rule):

```
{
      'op':'REPLY',
      'result':{
         'type':'121',
         'auth_type': '0',
         'auth_action': 'EDIT',
         'field' :'services',
         'old_value': [VALIDATOR],
         'new_value': []

         'reqId':441933878,
```

(continues on next page)

(continued from previous page)

```
          'identifier':'M9BJDuS24bqbJNvBRsoGg3',

          'data':[
               {
                 'auth_type': '0',
                 'auth_action': 'EDIT',
                 'field' :'services',
                 'old_value': [VALIDATOR],
                 'new_value': []
                 'constraint':{
                       'constraint_id': 'OR',
                       'auth_constraints': [{'constraint_id': 'ROLE',
                                             'role': '0',
                                             'sig_count': 2,
                                             'need_to_be_owner': False,
                                             'metadata': {}},

                                            {'constraint_id': 'ROLE',
                                             'role': '2',
                                             'sig_count': 1,
                                             'need_to_be_owner': True,
                                             'metadata': {}}
                                            ]
                 },
               }
          ],

          'state_proof':{
             'proof_nodes':
↪'+Pz4+pUgQURELS0xLS1yb2xlLS0qLS0xMDG44vjguN57ImF1dGhfY29uc3RyYWludHMiOlt7ImNvbnN0cmFpbnRfaWQiOiJST
↪',
             'root_hash':'DauPq3KR6QFnkaAgcfgoMvvWR6UTdHKZgzbjepqWaBqF',
             'multi_signature':{
                'signature':
↪'RNsPhUuPwwtA7NEf4VySCg1Fb2NpwapXrY8d64TLsRHR9rQ5ecGhRd89NTHabh8qEQ8Fs1XWawHjbSZ95RUYsJwx8PEXQcFEDG
↪',
                'value':{
                   'state_root_hash':'DauPq3KR6QFnkaAgcfgoMvvWR6UTdHKZgzbjepqWaBqF',
                   'pool_state_root_hash':'9L5CbxzhsNrZeGSJGVVpsC56JpuS5DGdUqfsFsR1RsFQ
↪',
                   'timestamp':1552395470,
                   'txn_root_hash':'4CowHvnk2Axy2HWcYmT8b88A1Sgk45x7yHAzNnxowN9h',
                   'ledger_id':2
                },
                'participants':[
                   'Beta',
                   'Gamma',
                   'Delta'
                ]
             }
          },
      }
}
```

Request Example (for getting all rules):

```
{
    'reqId':575407732,
    'signature':
↪'4AheMmtrfoHuAEtg5VsFPGe1j2w1UYxAvShRmfsCTSHnBDoA5EbmCa2xZzZVQjQGUFbYr65uznu1iUQhW22RNb1X
↪',
    'identifier':'M9BJDuS24bqbJNvBRsoGg3',
    'operation':{
        'type':'121'
    },
    'protocolVersion':2
}
```

Reply Example (for getting all rules):

```
{
      'op':'REPLY',
      'result':{
         'type':'121',

         'reqId':575407732,
         'identifier':'M9BJDuS24bqbJNvBRsoGg3'

         'data':[
              {
                'auth_type': '0',
                'auth_action': 'EDIT',
                'field' :'services',
                'old_value': [VALIDATOR],
                'new_value': []
                'constraint':{
                      'constraint_id': 'OR',
                      'auth_constraints': [{'constraint_id': 'ROLE',
                                            'role': '0',
                                            'sig_count': 2,
                                            'need_to_be_owner': False,
                                            'metadata': {}},

                                           {'constraint_id': 'ROLE',
                                            'role': '2',
                                            'sig_count': 1,
                                            'need_to_be_owner': True,
                                            'metadata': {}}
                                           ]
                },
              },
              {
                'auth_type': '102',
                'auth_action': 'ADD',
                'field' :'*',
                'new_value': '*'
                'constraint':{
                    'constraint_id': 'ROLE',
                    'role': '2',
                    'sig_count': 1,
                    'need_to_be_owner': False,
                    'metadata': {}
                },
```

(continues on next page)

(continued from previous page)

```
                },
                ........
            ],

        }
    }
```

2.6.9 GET_TRANSACTION_AUTHOR_AGREEMENT

Gets a transaction author agreement.

- Gets the latest (current) transaction author agreement if no input parameter is set.
- Gets a transaction author agreement by its digest if `digest` is set. The digest is calculated from concatenation of *TRANSACTION_AUTHOR_AGREEMENT*'s `version` and `text`.
- Gets a transaction author agreement by its version if `version` is set.
- Gets the latest (current) transaction author agreement at the given time (from ledger point of view) if `timestamp` is set.

All input parameters are optional and mutually exclusive.

- `digest` (sha256 digest hex string):

  Transaction's author agreement sha256 hash digest hex string calculated from concatenation of *TRANSACTION_AUTHOR_AGREEMENT*'s `version` and `text`.

- `version` (string):

  Unique version of the transaction author agreement.

- `timestamp` (integer as POSIX timestamp):

  The time when transaction author agreement has been ordered (written to the ledger).

Request Example:

```
{
    'operation': {
        'type': '6'
        'version': '1.0',
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        'type': '6',
        'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'version': '1.0',
```

(continues on next page)

(continued from previous page)

```
        'seqNo': 10,
        'txnTime': 1514214795,

        'data': {
            "version": "1.0",
            "text": "Please read carefully before writing anything to the ledger",
        },

        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7-
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 2,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtFI
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
            }
        },


    }
}
```

2.6.10 GET_TRANSACTION_AUTHOR_AGREEMENT_AML

Gets a transaction author agreement acceptance mechanisms list.

- Gets the latest (current) transaction author agreement acceptance mechanisms list if no input parameter is set.
- Gets a transaction author agreement acceptance mechanisms list by its version if `version` is set.
- Gets the latest (current) transaction author agreement acceptance mechanisms list at the given time (from ledger point of view) if `timestamp` is set.

All input parameters are optional and mutually exclusive.

- `version` (string):

  Unique version of the transaction author agreement acceptance mechanisms list

- `timestamp` (integer as POSIX timestamp):

  The time when transaction author agreement acceptance mechanisms list has been ordered (written to the ledger).

Request Example:

```
{
    'operation': {
        'type': '7'
        'version': '1.0',
    },

    'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
    'reqId': 1514308188474704,
    'protocolVersion': 2
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
        'type': '7',
        'identifier': 'L5AD5g65TDQr1PPHHRoiGf',
        'reqId': 1514308188474704,

        'version': '1.0',

        'seqNo': 10,
        'txnTime': 1514214795,

        'data': {
            "version": "1.0",
            "aml": {
                "EULA": "Included in the EULA for the product being used",
                "Service Agreement": "Included in the agreement with the service␣
↪provider managing the transaction",
                "Click Agreement": "Agreed through the UI at the time of submission",
                "Session Agreement": "Agreed at wallet instantiation or login"
            },
            "amlContext": "http://aml-context-descr"
        },

        'state_proof': {
            'root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH',
            'proof_nodes': '+QHl+FGAgICg0he/hjc9t/
↪tPFzmCrb2T+nHnN0cRwqPKqZEc3pw2iCaAoAsA80p3oFwfl4dDaKkNI8z8weRsSaS9Y8n3HoardRzxgICAgICAgICAgID4naAgw
↪CULIerYmmnnK2A6hN1u4ofU2eihKBna5MOCHiaObMfghjsZ8KBSbC6EpTFruD02fuGKlF1q4CAgICgBk8Cpc14mIr78WguSeT7-
↪rLT8qykKxzI4IO5ZMQwSmAoLsEwI+BkQFBiPsN8F610IjAg3+MVMbBjzugJKDo4NhYoFJ0ln1wq3FTWO0iw1zoUcO3FPjSh5ytv
↪',
            'multi_signature': {
                'value': {
                    'timestamp': 1514308168,
                    'ledger_id': 2,
                    'txn_root_hash': '4Y2DpBPSsgwd5CVE8Z2zZZKS4M6n9AbisT3jYvCYyC2y',
                    'pool_state_root_hash':
↪'9fzzkqU25JbgxycNYwUqKmM3LT8KsvUFkSSowD4pHpoK',
                    'state_root_hash': '81bGgr7FDSsf4ymdqaWzfnN86TETmkUKH4dj4AqnokrH'
                },
                'signature':
↪'REbtR8NvQy3dDRZLoTtzjHNx9ar65ttzk4jMqikwQiL1sPcHK4JAqrqVmhRLtw6Ed3iKuP4v8tgjA2BEvoyLTX6vB6vN4CqtFl
↪',
                'participants': ['Delta', 'Gamma', 'Alpha']
```

(continues on next page)

(continued from previous page)

```
            }
        },


    }
}
```

2.6.11 GET_TXN

A generic request to get a transaction from Ledger by its sequence number.

- `ledgerId` (int enum):

  ID of the ledger the requested transaction belongs to (Pool=0; Domain=1; Config=2).

- `data` (int):

  Requested transaction sequence number as it's stored on Ledger.

Request Example (requests a NYM txn with seqNo=9):

```
{
    'operation': {
        'type': '3',
        'ledgerId': 1,
        'data': 9
    },

    'identifier': 'MSjKTWkPLtYoPEaTF1TUDb',
    'reqId': 1514311281279625,
    'protocolVersion': 2
}
```

Reply Example (returns requested NYM txn with seqNo=9):

```
{
    "op": "REPLY",
    "result": {
        "type": "3",
        "identifier": "MSjKTWkPLtYoPEaTF1TUDb",
        "reqId": 1514311352551755,

        "seqNo": 9,

        "data": {
            "ver": 1,
            "txn": {
                "type":"1",
                "protocolVersion":2,

                "data": {
                    "ver": 1,
                    "dest":"GEzcdDLhCpGCYRHW82kjHd",
                    "verkey":"~HmUWn928bnFT6Ephf65YXv",
                    "role":101,
                },
```

(continues on next page)

(continued from previous page)

```
                "metadata": {
                    "reqId":1513945121191691,
                    "from":"L5AD5g65TDQr1PPHHRoiGf",
                    "digest":
↪"4ba05d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
                    "payloadDigest":
↪"21f0f5c158ed6ad49ff855baf09a2ef9b4ed1a8015ac24bccc2e0106cd905685",
                    "taaAcceptance": {
                        "taaDigest":
↪"6sh15d9b2c27e52aa8778708fb4b3e5d7001eecd02784d8e311d27b9090d9453",
                        "mechanism": "EULA",
                        "time": 1513942017
                     }
                },
            },
            "txnMetadata": {
                "txnTime":1513945121,
                "seqNo": 10,
                "txnId": "N22KY2Dyvmuu2PyyqSFKue|01"
            },
            "reqSignature": {
                "type": "ED25519",
                "values": [{
                    "from": "L5AD5g65TDQr1PPHHRoiGf",
                    "value":
↪"4X3skpoEK2DRgZxQ9PwuEvCJpL8JHdQ8X4HDDFyztgqE15DM2ZnkvrAh9bQY16egVinZTzwHqznmnkaFM4jjyDgd
↪"
                }]
            }

            "rootHash": "5ecipNPSztrk6X77fYPdepzFRUvLdqBuSqv4M9Mcv2Vn",
            "auditPath": ["Cdsoz17SVqPodKpe6xmY2ZgJ9UcywFDZTRgWSAYM96iA",
↪"3phchUcMsnKFk2eZmcySAWm2T5rnzZdEypW7A5SKi1Qt"],
        }

        "type": "3",
        "reqId": 1514311281279625,
        "identifier": "MSjKTWkPLtYoPEaTF1TUDb",
        "seqNo": 9,
    }
}
```

2.7 Action Requests

2.7.1 POOL_RESTART

The command to restart all nodes at the time specified in field "datetime"(sent by Trustee).

- `datetime` (string):

  Restart time in datetime frmat/ To restart as early as possible, send message without the "datetime" field or put in it value "0" or ""(empty string) or the past date on this place. The restart is performed immediately and there is no guarantee of receiving an answer with Reply.

- `action` (enum: `start` or `cancel`):

Starts or cancels the Restart.

Request Example:

```
{
    "reqId": 98262,
    "signature":
↪"cNAkmqSySHTckJg5rhtdyda3z1fQcV6ZVo1rvcd8mKmm7Fn4hnRChebts1ur7rGPrXeF1Q3B9N7PATYzwQNzdZZ
↪",
    "protocolVersion": 1,
    "identifier": "M9BJDuS24bqbJNvBRsoGg3",
    "operation": {
            "datetime": "2018-03-29T15:38:34.464106+00:00",
            "action": "start",
            "type": "118"
    }
}
```

Reply Example:

```
{
    "op": "REPLY",
    "result": {
            "reqId": 98262,
            "type": "118",
            "identifier": "M9BJDuS24bqbJNvBRsoGg3",
            "datetime": "2018-03-29T15:38:34.464106+00:00",
            "action": "start",
    }
}
```

2.7.2 VALIDATOR_INFO

Command provide info from all the connected nodes without need of consensus.

Request Example:

```
{
    'protocolVersion': 2,
    'reqId': 83193,
    'identifier': 'M9BJDuS24bqbJNvBRsoGg3',
    'operation': {
                'type': '119'
    }
}
```

Reply Example:

```
{
    'op': 'REPLY',
    'result': {
            'reqId': 83193,
            'data': { <Json with node info> },
            'type': '119',
            'identifier': 'M9BJDuS24bqbJNvBRsoGg3'
    }
}
```

CHAPTER 3

Default AUTH_MAP Rules

3.1 Who Is Owner

CHAPTER 4

Pool Upgrade Guideline

There is quite interesting and automated process of the pool (network) upgrade.

- The whole pool (that is each node in the pool) can be upgraded automatically without any manual actions via `POOL_UPGRADE` transaction.
- As a result of Upgrade, each Node will be at the specified version, that is a new package, for example deb package, will be installed.
- Migration scripts can also be performed during Upgrade to deal with breaking changes between the versions.

4.1 Pool Upgrade Transaction

- Pool Upgrade is done via `POOL_UPGRADE` transaction.
- The txn defines a schedule of Upgrade (upgrade time) for each node in the pool.
- Only the `TRUSTEE` can send `POOL_UPGRADE`.
- This is a common transaction (written to config ledger), so consensus is required.
- There are two main modes for `POOL_UPGRADE`: forced and non-forced (default).
  - Non-forced mode schedules upgrade only after `POOL_UPGRADE` transaction is written to the ledger, that is there was consensus. Forced upgrade schedules upgrade for each node regardless of whether `POOL_UPGRADE` transaction is actually written to the ledger, that is it can be scheduled even if the pool lost consensus.
  - Non-forced mode requires that upgrade of each node is done sequentially and not at the same time (so that a pool is still working and can reach consensus during upgrade). Forced upgrade allows upgrade of the whole pool at the same time.
- One should usually use non-forced Upgrades assuming that all changes are backward-compatible.
- If there are non-backward-compatible (breaking) changes, then one needs to use forced Upgrade and make it happen at the same time on all nodes (see below).

4.2 Node Upgrade Transaction

- Each node sends `NODE_UPGRADE` transaction twice:
  - `in_progress` action: just before start of the Upgrade (that is re-starting the node and applying a new package) to log that Upgrade started on the node.
  - `success` or `fail` action: after upgrade of the node to log the upgrade result.
- `NODE_UPGRADE` transaction is a common transaction (written to config ledger), so consensus is required.

4.3 Node Control Tool

- Upgrade is performed by a `node-control-tool`.
- See `node_control_tool.py`.
- On Ubuntu it's installed as a systemd service (`indy-node-control`) in addition to the node service (`indy-node`).
- `indy-node-control` is executed from the `root` user.
- When upgrade time for the node comes, it sends a message to node-control-tool.
- The node-control-tool then does the following:
  - stops `indy-node` service;
  - upgrades `indy-node` package (`apt-get install` on Ubuntu);
  - back-ups node data (ledger, etc.);
  - runs migration scripts (see `migration_tool.py`);
  - starts `indy-node` service;
  - restarts `indy-node-control` service.
- If upgrade failed for some reasons, node-control-tool tries to restore the data (ledger) from back-up and revert to the version of the code before upgrade.

4.4 Migrations

- Although we must try keeping backward-compatibility between the versions, it may be possible that there are some (for example, changes in ledger and state data format, re-branding, etc.).
- We can write migration scripts to support this kind of breaking changes and perform necessary steps for data migration and/or running some scripts.
- The migration should go to `data/migration` folder under the package name (so this is `data/migration/deb` on Ubuntu).
- Please have a look at the following doc for more information about how to write migration scripts.

4.5 When to Run Forced Upgrades

- Any changes in Ledger transactions format leading to changes in transactions root hash.
- Any changes in State transactions format (for example new fields added to State values) requiring re-creation of the state from the ledger.
- Any changes in Requests/Replies/Messages without compatibility and versioning support.

CHAPTER 5

Create a Network and Start Nodes

Please be aware that recommended way of starting a pool is to use Docker.

In order to run your own Network, you need to do the following for each Node:

1. Install Indy Node

   - A recommended way for ubuntu is installing from deb packages

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys CE7709D068DB5E88
sudo bash -c 'echo "deb https://repo.sovrin.org/deb xenial stable" >> /etc/apt/
↪sources.list'
sudo apt-get update
sudo apt-get install indy-node
```

   - It's also possible to install from pypi for test purposes: `pip install indy-node`

2. Initialize Node to be included into the Network

   - if `indy-node` were installed from pypi basic directory structure should created manually with the command `create_dirs.sh`
   - set Network name in config file
     - the location of the config depends on how a Node was installed. It's usually inside `/etc/indy` for Ubuntu.
     - the following needs to be added: `NETWORK_NAME={network_name}` where {network_name} matches the one in genesis transaction files above
   - generate keys
     - ed25519 transport keys (used by ZMQ for Node-to-Node and Node-to-Client communication)
     - BLS keys for BLS multi-signature and state proofs support
   - provide genesis transactions files which will be a basis of initial Pool.
     - pool transactions genesis file:
       * The file must be named as `pool_transactions_genesis`

    * The file contains initial set of Nodes a Pool is started from (initial set of NODE transactions in the Ledger)
    * New Nodes will be added by sending new NODE txn to be written into the Ledger
    * All new Nodes and Clients will use genesis transaction file to connect to initial set of Nodes, and then catch-up all other NODE transactions to get up-to-date Ledger.
    * File must be located in `/var/lib/indy/{network_name}` folder
  - domain transactions genesis file:
    * The file must be named as `domain_transactions_genesis`
    * The file contains initial NYM transactions (for example, Trustees, Stewards, etc.)
    * File must be located in `/var/lib/indy/{network_name}` folder
- configure iptables to limit the number of simultaneous clients connections (recommended)

5.1 Scripts for Initialization

There are a number of scripts which can help in generation of keys and running a test network.

5.1.1 Generating keys

For deb installation

The following script should be used to generate both ed25519 and BLS keys for a node named `Alpha` with node port `9701` and client port `9702`:

```
init_indy_node Alpha 0.0.0.0 9701 0.0.0.0 9702 [--seed␣
↪111111111111111111111111111Alpha]
```

Also this script generates indy-node environment file needed for systemd service config and indy-node iptables setup script.

For pip installation

The following script can generate both ed25519 and BLS keys for a node named `Alpha`:

```
init_indy_keys --name Alpha [--seed 111111111111111111111111111Alpha] [--force]
```

Note: Seed can be any randomly chosen 32 byte value. It does not have to be in the format 11..

Please note that these scripts must be called *after* CURRENT_NETWORK is set in config (see above).

5.1.2 Generating keys and test genesis transaction files for a test network

There is a script that can generate keys and corresponding test genesis files to be used with a Test network.

```
~$ generate_indy_pool_transactions --nodes 4 --clients 5 --nodeNum 1 [--ips '191.177.
↪76.26,22.185.194.102,247.81.153.79,93.125.199.45'] [--network=sandbox]
```

- `--nodes` specifies a total number of nodes in the pool

- `--clients` specifies a number of pre-configured clients in the pool (in `domain_transactions_file_{network_name}_genesis`)
- `--nodeNum` specifies a number of this particular node (from 1 to `-nodes` value), that is a number of the Node to create private keys locally for
- `--ip` specifies IP addresses for all nodes in the pool (if not specified, then `localhost` is used)
- `--network` specifies a Network generate transaction files and keys for. `sandbox` is used by default

We can run the script multiple times for different networks.

5.1.3 Setup iptables (recommended)

It is strongly recommended to add iptables (or some other firewall) rule that limits the number of simultaneous clients connections for client port. There are at least two important reasons for this:

- preventing the indy-node process from reaching of open file descriptors limit caused by clients connections
- preventing the indy-node process from large memory usage as ZeroMQ creates the separate queue for each TCP connection.

Instructions related to iptables setup can be found here.

5.1.4 Running Node

The following script will start a Node process which can communicate with other Nodes and Clients:

```
start_indy_node Alpha 0.0.0.0 9701 0.0.0.0 9702
```

The node uses separate TCP channels for communicating with nodes and clients. The first IP/port pair is for the node-to-node communication channel and the second IP/port pair is for node-to-client communication channel. IP addresses may be changed according to hardware configuration. Different IP addresses for node-to-node and node-to-client communication may be used.

5.2 Local Test Network Example

If you want to try out an Indy cluster of 4 nodes with the nodes running on your local machine, then you can do the following:

```
~$ generate_indy_pool_transactions --nodes 4 --clients 5 --nodeNum 1 2 3 4
By default node with the name Node1 will use ports 9701 and 9702 for nodestack and␣
↪clientstack respectively
Node2 will use ports 9703 and 9704 for nodestack and clientstack respectively
Node3 will use ports 9705 and 9706 for nodestack and clientstack respectively
Node4 will use ports 9707 and 9708 for nodestack and clientstack respectively
```

Now you can run the 4 nodes as

```
start_indy_node Node1 0.0.0.0 9701 0.0.0.0 9702
```

```
start_indy_node Node2 0.0.0.0 9703 0.0.0.0 9704
```

```
start_indy_node Node3 0.0.0.0 9705 0.0.0.0 9706
start_indy_node Node4 0.0.0.0 9707 0.0.0.0 9708
```

5.3 Remote Test Network Example

Now let's say you want to run 4 nodes on 4 different machines as

1. Node1 running on 191.177.76.26
2. Node2 running on 22.185.194.102
3. Node3 running on 247.81.153.79
4. Node4 running on 93.125.199.45

On machine with IP 191.177.76.26 you will run

```
~$ generate_indy_pool_transactions --nodes 4 --clients 5 --nodeNum 1 --ips '191.177.
↪76.26,22.185.194.102,247.81.153.79,93.125.199.45'
This node with name Node1 will use ports 9701 and 9702 for nodestack and clientstack␣
↪respectively
```

On machine with IP 22.185.194.102 you will run

```
~$ generate_indy_pool_transactions --nodes 4 --clients 5 --nodeNum 2 --ips '191.177.
↪76.26,22.185.194.102,247.81.153.79,93.125.199.45'
This node with name Node2 will use ports 9703 and 9704 for nodestack and clientstack␣
↪respectively
```

On machine with IP 247.81.153.79 you will run

```
~$ generate_indy_pool_transactions --nodes 4 --clients 5 --nodeNum 3 --ips '191.177.
↪76.26,22.185.194.102,247.81.153.79,93.125.199.45'
This node with name Node3 will use ports 9705 and 9706 for nodestack and clientstack␣
↪respectively
```

On machine with IP 93.125.199.45 you will run

```
~$ generate_indy_pool_transactions --nodes 4 --clients 5 --nodeNum 4 --ips '191.177.
↪76.26,22.185.194.102,247.81.153.79,93.125.199.45'
This node with name Node4 will use ports 9707 and 9708 for nodestack and clientstack␣
↪respectively
```

Now you can run the 4 nodes as

```
start_indy_node Node1 0.0.0.0 9701 0.0.0.0 9702
```

```
start_indy_node Node2 0.0.0.0 9703 0.0.0.0 9704
```

```
start_indy_node Node3 0.0.0.0 9705 0.0.0.0 9706
```

```
start_indy_node Node4 0.0.0.0 9707 0.0.0.0 9708
```

CHAPTER 6

Add Node to Existing Pool

6.1 Node preparation

1. Add this files from a running node:

```
/var/lib/indy/network_name/pool_transactions_genesis
/var/lib/indy/network_name/domain_transactions_genesis
```

1. Initialize keys, aliases and ports on the new node using init_indy_node script. Example:

```
sudo su - indy -c "init_indy_node NewNode 0.0.0.0 9701 0.0.0.0 9702␣
↪000000000000000000000000NewNode"
```

1. When the node starts for the first time, it reads the content of genesis `pool_transactions_sandbox` and `domain_transactions_sandbox` files and adds it to the ledger. The Node reads genesis transactions only once during the first start-up, so make sure the genesis files are correct before starting the service.

```
sudo systemctl start indy-node
sudo systemctl status indy-node
sudo systemctl enable indy-node
```

6.2 Add Node to the Pool

1. As Trustee add another Steward if needed (only Steward can add a new Validator Node; a Steward can add one and only one Validator Node).
2. Using Indy CLI, run the following command as Steward:

```
ledger node target=6G9QhQa3HWjRKeRmEvEkLbWWf2t7cw6KLtafzi494G4G client_port=9702␣
↪client_ip=10.255.255.255 alias=NewNode node_ip=10.0.0.10.255.255.255 node_port=9701␣
↪services=VALIDATOR␣
↪blskey=zi65fRHZjK2R8wdJfDzeWVgcf9imXUsMSEY64LQ4HyhDMsSn3Br1vhnwXHE7NyGjxVnwx4FGPqxpzY8HrQ2PnrL9tu4u
↪blskey_
↪pop=RaY9xGLbQbrBh8np5gWWQAWisaxd96FtvbxKjyzBj4fUYyPq4pkyCHTYvQzjehmUK5pNfnyhwWqGg1ahPwtWopenuRjAeCk
```

(continues on next page)

(continued from previous page)

- `alias` specifies unique Node name
- `blskey` specifies BLS key from `init_indy_node` script
- `blskey_pop` specifies Proof of possession for BLS key from `init_indy_node` script
- `target` specifies base58 of the node public key ('Verification key' field in output of `init_indy_node`)

Example: Verification key is `ab78300b3a3eca0a1679e72dd1656075de9638ae79dc6469a3093ce1cc8b424f`In order to get base58 of the verkey execute in your shell (you should have `indy-plenum` installed):`python3 -c "from plenum.common.test_network_setup import TestNetworkSetup; print(TestNetworkSetup.getNymFromVerkey(str.encode('ab78300b3a3eca0a1679e72dd1656075de9638ae79dc6469a3093ce1cc8b424f')))"`**Output:**

4Tn3wZMNCvhSTXPcLinQDnHyj56DTLQtL61ki4jo2Loc

6.3 Make sure that Node is workable

Do `systemctl restart indy-node` and verify that node completed catch-up successfully.

CHAPTER 7

Helper Scripts

Indy-node comes with a number of useful helper scripts:

- `init_indy_keys`

  Initializes (or rotates) Node's keys (private and public ones) needed for communication with the Nodes via CurveCP protocol (CurveZMQ)

- `init_bls_keys`

  Initializes (or rotates) Node's BLS keys required for BLS multi-signature and State Proofs support

- `read_ledger`

  Reads transaction from a specified ledger in JSON format

- `validator-info`

  Monitors the current status of the Node

- `generate_indy_pool_transactions`

  Initializes a test Pool (generates keys and genesis transactions)

- `clear_node`

  Clean up of all data on the Node

CHAPTER 8

Setup iptables rules (recommended)

It is strongly recommended to add iptables (or some other firewall) rule that limits the number of simultaneous clients connections for client port. There are at least two important reasons for this:

- preventing the indy-node process from reaching of open file descriptors limit caused by clients connections
- preventing the indy-node process from large memory usage as ZeroMQ creates the separate queue for each TCP connection.

NOTE: limitation of the number of *simultaneous clients connections* does not mean that we limit the number of *simultaneous clients* the indy-node works with in any time. The IndySDK client does not keep connection infinitely, it uses the same connection for request-response session with some optimisations, so it's just about **connections**, **not** about **clients**.

Also iptables can be used to deal with various DoS attacks (e.g. syn flood) but rules' parameters are not estimated yet.

NOTE: you should be a root to operate with iptables.

8.1 Setting up clients connections limit

8.1.1 Using raw iptables command or iptables front-end

In case of deb installation the indy-node environment file /etc/indy/indy.env is created by `init_indy_node` script. This environment file contains client port (NODE_CLIENT_PORT) and recommended clients connections limit (CLIENT_CONNECTIONS_LIMIT). This parameters can be used to add the iptables rule for chain INPUT:

```
# iptables -I INPUT -p tcp --syn --dport 9702 -m connlimit --connlimit-above 500 --
↪connlimit-mask 0 -j REJECT --reject-with tcp-reset
```

Some key options:

- –dport - a port for which limit is set
- –connlimit-above - connections limit, exceeding new connections will be rejected using TCP reset
- –connlimit-mask - group hosts using the prefix length, 0 means “all subnets”

Corresponding fields should be set in case of some iptables front-end usage.

8.1.2 Using indy scripts

For ease of use and for people that are not familiar with iptables we've added two scripts:

- setup_iptables: adds a rule to iptables to limit the number of simultaneous clients connections for specified port;
- setup_indy_node_iptables: a wrapper for setup_iptables script which gets client port and recommended connections limit from indy-node environment file that is created by init_indy_node script.

Links to these scripts:

- https://github.com/hyperledger/indy-node/blob/master/scripts/setup_iptables
- https://github.com/hyperledger/indy-node/blob/master/scripts/setup_indy_node_iptables

NOTE: for now the iptables chain for which the rule is added is not parameterized, the rule is always added for INPUT chain, we can parameterize it in future if needed.

For deb installation

To setup the limit of the number of simultaneous clients connections it is enough to run the following script without parameters

```
# setup_indy_node_iptables
```

This script gets client port and recommended connections limit from the indy-node environment file.

NOTE: this script should be called *after* `init_indy_node` script.

For pip installation

The `setup_indy_node_iptables` script can not be used in case of pip installation as indy-node environment file does not exist, use the `setup_iptables` script instead (9702 is a client port, 500 is recommended limit for now)

```
# setup_iptables 9702 500
```

In fact, the `setup_indy_node_iptables` script is just a wrapper for the `setup_iptables` script.

CHAPTER 9

Node Monitoring Tools for Stewards

9.1 Plugin Manager

Currently, indy-node emits different events via the Plugin Manager when certain criteria are met. The Plugin Manager tries to import all pip packages which names start with "indynotifier*". Each of these packages is required to expose `send_message`; interface which is used to pass the event with the associated message to the package for further handling.

The Plugin Manager code is located at here.

9.1.1 Events Emitted

- .nodeRequestSpike : NodeRequestSuspiciousSpike
- .clusterThroughputSpike : ClusterThroughputSuspiciousSpike
- .clusterLatencyTooHigh : ClusterLatencyTooHigh
- .nodeUpgradeScheduled : NodeUpgradeScheduled
- .nodeUpgradeComplete : NodeUpgradeComplete
- .nodeUpgradeFail : NodeUpgradeFail
- .poolUpgradeCancel : PoolUpgradeCancel

9.2 Email Plugin

9.2.1 Prerequisites

- SMTP server must be running on localhost.
- Install SMTP server (if you don't have one already)

The most simple way on Ubuntu is to use `sendmail`:

```
$ sudo apt-get install sendmail
```

To check that it's working execute:

```
echo "Subject:  sendmail test" | sendmail -v youremail@example.com -f
alert@noreply.com
```

If you get a email on your youremail@example.com then `sendmail` is working.

9.2.2 Install

```
# pip3 install indynotifieremail
```

9.2.3 Configuration

The spike detection and notification mechanisms should be enabled by appending of the following line to `indy_config.py` configuration file:

```
SpikeEventsEnabled=True
```

The package depends on two environment variables:

- `INDY_NOTIFIER_EMAIL_RECIPIENTS` (required)
- `INDY_NOTIFIER_EMAIL_SENDER` (optional)

Add these variables to `/etc/indy/indy.env` environment file as you are required to set such system environment variables for indy-node service in form described below.

INDY_NOTIFIER_EMAIL_RECIPIENTS

`INDY_NOTIFIER_EMAIL_RECIPIENTS` should be a string in a format of:

`recipient1@adress.com [optional list of events the recipient is going to get], recipient2@adress.com [event list]`

If no list was provided the recipient is going to get notifications for all events. Example:

`steward1@company.com event1 event2, steward2@company.com, steward3@company.com event3`

This way steward1 is going to get notifications for event1 and event2, steward2 is going to get all possible notifications and steward3 is going to get notifications for event3 only.

The current list of events can be found above.

INDY_NOTIFIER_EMAIL_SENDER

By default every email notification is going to be from alert@noreply.com. You can change this by setting `INDY_NOTIFIER_EMAIL_SENDER`. May be useful for email filters.

9.2.4 Email delivery frequency

By default you will not get a email with the same topic more than once an hour. This is defined by `SILENCE_TIMEOUT`. It can be overridden by setting `INDY_NOTIFIER_SILENCE_TIMEOUT` environment variable in `/etc/indy/indy.env` file. Emails regarding update procedure are always delivered.

9.3 AWS SNS Plugin

9.3.1 Prerequisites

- .A AWS SNS topic created with permissions to publish to it.
- .A installed Sovrin Validator instance.

9.3.2 Setup

Install the python package for sovrin-notifier-awssns. This should be only be installed using pip3.

`pip3 install sovrinnotifierawssns`

9.3.3 Configuration

To configure AWS Credentials you will need to know the values for: `aws_access_key_id` and `aws_secret_access_key`. Follow the steps documented here Boto3 Configuring Credentials.

Use either of the following ways:

- .Environment variables `AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY`
- .Shared credential file (~/.aws/credentials)
- .Boto2 config file (/etc/boto.cfg and ~/.boto)

Configure AWS Region you will need to know the value where the SNS Topic is hosted e.g. us-west-1, us-west-2, sa-east-1

To achieve this:

- .Set a Environment variable AWS_DEFAULT_REGION

- .Set region using file (~/.aws/config)

Define environment variable `SOVRIN_NOTIFIER_AWSSNS_TOPICARN` on the Validator and set valid AWS SNS TopicARN as the value.

9.3.4 Events

Events that cause a notification:

- `NodeRequestSuspiciousSpike`
- `ClusterThroughputSuspiciousSpike`
- `ClusterLatencyTooHigh`
- `NodeUpgradeScheduled`
- `NodeUpgradeComplete`
- `NodeUpgradeFail,`
- `PoolUpgradeCancel`

9.3.5 Hints

The home directory for the account that runs `sovrin-node.service` on a Validator is `/home/sovrin/`. So the aws credentials/config files must be created in `/home/sovrin/.aws` folder.

To set an environment variable on the Validator you must add it to the file `/home/sovrin/.sovrin/sovrin.env` and restart the Validator. The TopicARN must be defined in this file.

To restart the Validator on a Ubuntu system you must execute the command `sudo systemctl restart sovrin-node.service` while not logged in as a sovrin user.

Example

This simple script will complete the setup, assuming that the sovrinnotifierawssns package is already installed:

```
#!/bin/bash

sudo mkdir /home/sovrin/.aws

sudo sh -c "printf \"[default]\nregion=us-west-2\" > /home/sovrin/.aws/config"

sudo sh -c "printf \" .[default]\naws_access_key_id=AKIAIGKGW3CKRXKKWPZA\naws_secret_
↪access_key=<YOUR_SECRET_KEY>\" > /home/sovrin/.aws/credentials"

sudo sh -c "printf \"SOVRIN_NOTIFIER_AWSSNS_TOPICARN=arn:aws:sns:us-west-
↪2:034727365312:validator-health-monitor-STN\" >> /home/sovrin/.sovrin/sovrin.env"

sudo chown -R sovrin:sovrin /home/sovrin/.aws /home/sovrin/.sovrin/sovrin.env

sudo systemctl restart sovrin-node
```

CHAPTER 10

Continuous Integration / Delivery

10.1 Branches

- `master` branch contains the latest changes. All PRs usually need to be sent to master.
- `stable` branch contains latest releases (https://github.com/hyperledger/indy-node/releases). Hotfixes need to be sent to both stable and master.
- `release-*` branches hold release candidates during release workflow
- `hotfix-*` branches hold release candidates during hotfix workflow

10.2 Pull Requests

- Each PR needs to be reviewed.
- PR can be merged only after all tests pass and code is reviewed.

10.3 Continuous Integration

- for each PR we execute:
  - static code validation
  - Unit/Integration tests
- We use pipeline in code approach and Jenkins as our main CI/CD server.
- CI part of the pipeline (running tests for each PR) is defined in `Jenkinsfile.ci` file.
- CI part is run on Hyperledger and Sovrin Foundation Jenkins servers, so they are public and open as every contributor needs to see results of the tests run for his or her PR.

10.3.1 Static Code Validation

- We use flake8 for static code validation.
- It's run against every PR. PR fails if there are some static code validation errors.
- Not all checks are enabled (have a look at `.flake8` file at the project root)
- You can run static code validation locally:
  - Install flake8: `pip install flake8`
  - Run validation on the root folder of the project: `flake8 .`

10.4 Continuous Delivery

- CD part of the pipeline is defined in `Jenkinsfile.cd` file.
- CD part is run on Sovrin Foundation Jenkins server dealing with issuing and uploading new builds.

10.4.1 Builds

What artifacts are produced after each push

- to `master` branch:
  - all artifacts include developmental release segment `devN` in their version
  - indy-plenum:
    - indy-plenum in pypi
    - indy-plenum deb package in `https://repo.sovrin.org/deb xenial master-latest`
  - indy-node:
    - indy-node in pypi
    - indy-node deb package in `https://repo.sovrin.org/deb xenial master-latest`
    - indy-node deb package in `https://repo.sovrin.org/deb xenial master` (copied from `master-latest`)
    - indy-plenum deb package in `https://repo.sovrin.org/deb xenial master` (copied from `master-latest`)
- to `release-*` and `hotfix-*` branches:
  - all artifacts include pre-release segment `rcN` in their version
  - indy-plenum:
    - indy-plenum in pypi
    - indy-plenum deb package in `https://repo.sovrin.org/deb xenial rc-latest`
  - indy-node:
    - indy-node in pypi
    - indy-node deb package in `https://repo.sovrin.org/deb xenial rc-latest`
    - indy-node deb package in `https://repo.sovrin.org/deb xenial rc` (copied from `rc-latest`)

    * indy-plenum deb package in `https://repo.sovrin.org/deb xenial rc` (copied from `rc-latest`)

- to `stable` branch:
  - indy-plenum:
    * indy-plenum in pypi
    * indy-plenum deb package in `https://repo.sovrin.org/deb xenial stable-latest`
    * indy-plenum release tag (https://github.com/hyperledger/indy-plenum/releases)
  - indy-node:
    * indy-node in pypi
    * indy-node deb package in `https://repo.sovrin.org/deb xenial stable-latest` (re-packed from `rc-latest`)
    * indy-node deb package in `https://repo.sovrin.org/deb xenial stable` (copied from `rc-latest`)
    * indy-plenum deb package in `https://repo.sovrin.org/deb xenial stable` (copied from `stable-latest`)
    * indy-node release tag (https://github.com/hyperledger/indy-node/releases)

Use cases for artifacts

- PyPI artifacts can be used for development experiments, but not intended to be used for production.
- Using deb packages is recommended way to be used for a test/production pool on Ubuntu.
  - indy-node deb package from `https://repo.sovrin.org/deb xenial stable` is one and the only official stable release that can be used for production (stable version).
  - indy-node deb package from `https://repo.sovrin.org/deb xenial master` contains the latest changes (from master branch). It's not guaranteed that that this code is stable enough.

10.4.2 Packaging

Supported platforms and OSes

- Ubuntu 16.04 on x86_64

Build scripts

We use fpm for packaging python code into deb packages. Build scripts are placed in `build-scripts` folders:

- https://github.com/hyperledger/indy-node/blob/master/build-scripts
- https://github.com/hyperledger/indy-plenum/blob/master/build-scripts

We also pack some 3rd parties dependencies which are not presented in canonical ubuntu repositories:

- https://github.com/hyperledger/indy-node/blob/master/build-scripts/ubuntu-1604/build-3rd-parties.sh
- https://github.com/hyperledger/indy-plenum/blob/master/build-scripts/ubuntu-1604/build-3rd-parties.sh

Each `build-scripts` folder includes `Readme.md`. Please check them for more details.

10.4.3 Versioning

- Please note, that we are using versioning that satisfies PEP 440 with release segment as `MAJOR.MINOR.PATCH` that satisfies SemVer as well.
- Version is set in the code (see __version__.json).
- Version is bumped for new releases / hotfixes either manually or using bump_version.sh script. The latter is preferred.
- During development phase version includes developmental segment `devN`, where `N` is set for CD pipeline artifacts as incremented build number of build server jobs. In the source code it is just equal to `0` always.
- During release preparation phase (release / hotfix workflows) version includes pre-release segment `rcN`, where `N>=1` and set in the source code by developers.
- Each dependency (including indy-plenum) has a strict version (see setup.py)
- If you install indy-node (either from pypi, or from deb package), the specified in setup.py version of indy-plenum is installed.
- Master and Stable share the same versioning scheme.
- Differences in master and stable code:
  - `setup.py`: different versions of indy-plenum dependency
  - different versions in migrations scripts

For releases < `1.7.0` (deprecated)

- Please note, that we are using semver-like approach for versioning (major, minor, build) for each of the components.
- Major and minor parts are set in the code (see __metadata__.py). They must be incremented for new releases manually from code if needed.
- Build part is incremented with each build on Jenkins (so it always increases, but may be not sequentially)
- Each dependency (including indy-plenum) has a strict version (see setup.py)
- If you install indy-node (either from pypi, or from deb package), the specified in setup.py version of indy-plenum is installed.
- Master and Stable builds usually have different versions.
- Differences in master and stable code:
  - `setup.py`:
    - dev suffix in project names and indy-plenum dependency in master; no suffixes in stable
    - different versions of indy-plenum dependency
  - different versions in migrations scripts

10.5 Release workflow

10.5.1 Feature Release

1. Release Candidate Preparation

1. [**Maintainer**]
    - Create `release-X.Y.Z` branch from `stable` (during the first RC preparation only).
2. [**Contributor**]
    - Create `rc-X.Y.Z.rcN` branch from `release-X.Y.Z` (`N` starts from `1` and is incremented for each new RC).
    - Apply necessary changes from `master` (either `merge` or `cherry-pick`).
    - (*optional*) `[indy-node]` Set `indy-plenum` version in `setup.py`.
    - Set the package version `./bump_version.sh X.Y.Z.rcN`.
    - Commit, push and create a PR to `release-X.Y.Z`.
3. Until PR is merged:
    1. [**build server**]
        - Run CI for the PR and notifies GitHub.
    2. [**Maintainer**]
        - Review the PR.
        - Either ask for changes or merge.
    3. [**Contributor**]
        - (*optional*) Update the PR if either CI failed or reviewer asked for changes.
        - (*optional*) [**indy-node**] Bump `indy-plenum` version in `setup.py` if changes require new `indy-plenum` release.

2. Release Candidate Acceptance

Note If any of the following steps fails new release candidate should be prepared.

1. [**Maintainer**]
    - **Start release candidate pipeline manually**.
2. [**build server**]
    - Checkout the repository.
    - Publish to PyPI as `X.Y.Z.rcN`.
    - Bump version locally to `X.Y.Z`, commit and push as the `release commit` to remote.
    - Build debian packages:
        - for the project: source code version would be `X.Y.Z`, debian package version `X.Y.Z~rcN`;
        - for the 3rd party dependencies missed in the official debian repositories.
    - Publish the packages to `rc-latest` debian channel.
    - `[indy-node]` Copy the package along with its dependencies (including `indy-plenum`) from `rc-latest` to `rc` channel.
    - `[indy-node]` Run system tests for the `rc` channel.
    - Create **release PR** from `release-X.Y.Z` (that points to `release commit`) branch to `stable`.

- Notify maintainers.
- Wait for an approval to proceed. **It shouldn't be provided until `release PR` passes all necessary checks** (e.g. DCO, CI testing, maintainers reviews etc.).

3. [**build server**]
   - Run CI for the PR and notify GitHub.
4. [**QA**]
   - (*optional*) Perform additional testing.
5. [**Maintainer**]
   - Review the PR but **do not merge it**.
   - If approved: let build server to proceed.
   - Otherwise: stop the pipeline.
6. [**build server**]
   - If approved:
     - perform fast-forward merge;
     - create and push tag `vX.Y.Z`;
     - Notify maintainers.
   - Otherwise rollback `release commit` by moving `release-X.Y.Z` to its parent.

3. Publishing

1. [**build server**] triggered once the `release PR` is merged
   - Publish to PyPI as `X.Y.Z`.
   - Download and re-pack debian package `X.Y.Z~rcN` (from `rc-latest` channel) to `X.Y.Z` changing only the package name.
   - Publish the package to `rc-latest` debian channel.
   - Copy the package along with its dependencies from `rc-latest` to `stable-latest` channel.
   - [`indy-node`] Copy the package along with its dependencies (including `indy-plenum`) from `stable-latest` to `stable` channel.
   - [`indy-node`] Run system tests for the `stable` channel.
   - Notify maintainers.

4. New Development Cycle Start

1. [**Contributor**]:
   - Create PR to `master` with version bump to `X'.Y'.Z'.dev0`, where `X'.Y'.Z'` is next target release version. Usually it increments one of `X`, `Y` or `Z` and resets lower parts (check SemVer for more details), e.g.:
     - `X.Y.Z+1` - bugfix release
     - `X.Y+1.0` - feature release, backwards compatible API additions/changes
     - `X+1.0.0` - major release, backwards incompatible API changes

10.5.2 Hotfix Release

Hotfix release is quite similar except the following difference:

- hotfix branches named `hotfix-X.Y.Z`;
- `master` usually is not merged since hotfixes (as a rule) should include only fixes for stable code.

CHAPTER 11

Indy File Folders Structure Guideline

Indy-node service works with some files and folders on the file system. We need to be careful when selecting this files and folders or adding new ones.

11.1 1. Use system-specific files and folder for indy-node service

As of now, we support indy-node service (using systemctl) on Ubuntu only. But in future we will support more platforms (CentOS, Windows Server, etc.)

So, we should follow the following principles:

- Use system-specific folder for storing config files
  - Indy-config files
  - other config files (such as service config)

  Ubuntu: /etc/indy
- Use system-specific folder for storing data, such as
  - ledger (transaction log, states)
  - Genesis transaction files
  - Node keys (transport and BLS)

  Ubuntu: `/var/lib/indy`
- Use system-specific folder to store log files

  Ubuntu: `/var/log/indy`
- Avoid using /home folder for indy-node service

11.2 2. Organize file folders to support possibility to work with multiple networks (live, test, local, etc.)

We may have multiple Networks (identified by different genesis transaction files) installed for the same indy-node service. The file structure should support it.

- The current Network to work with is specified explicitly in the main config file (`NETWORK_NAME=`):

  Ubuntu: /etc/indy

- Separate config files for each Network

  Ubuntu: `/var/lib/indy/{network_name}`

- Separate data for each Network

  - Separate ledgers (transaction log, states)
  - Separate genesis transaction files
  - Separate Node keys (transport and BLS)

  Ubuntu: `/var/lib/indy/{network_name}`

- Separate log files for each Network

  Ubuntu: `/var/log/indy/{network_name}`

11.3 3. Set proper permissions for files and folders

Make sure that all data, and, especially, keys have proper permissions. Private keys can only be read by the user the service is run for (indy user usually)

11.4 4. Provide a way to override config and other data for different networks

Each Network may have its own config extending base config.

Ubuntu:

- `/etc/indy` - base config
- `/etc/indy/{network_name}` - config extension

11.5 5. Client should use /home folder

Client doesn't need a service, and should use its own home directory for files with proper permissions.

Indy-sdk uses `~/.indy_client`

11.6 6. Separate node and client folders

Client and Node should be two independent products not sharing any common files/folders

11.7 7. It should be possible to work with both node and client (libindy) on the same machine

We may install and work with both node and client on the same machine independently

11.8 8. It should be possible to run client (libindy) for multiple users

We may have multiple users working with client code on the same machine. Each user must have separate data and key files with proper permissions.

CHAPTER 12

Code quality requirements guideline

Please make sure that you take into account the following items before sending a PR with the new code:

12.1 General items

- Consider sending a design doc into `design` folder (as markdown or PlantUML diagram) for a new feature before implementing it.
- Make sure that a new feature or fix is covered by tests (try following TDD)
- Make sure that documentation is updated according to your changes (see docs folder). In particular, update *transactions* and *requests* if you change any transactions or requests format.
- Follow incremental re-factoring approach:
  - do not hesitate to improve the code
  - put TODO and FIXME comments if you see an issue in the code
  - log tickets in Indy Jira if you see an issue.

12.2 Code quality items

The current code is not perfect, so feel free to improve it.

- Follow PEP 8 and keep same code style in all files.
  - Naming. Snake case for functions, camel case for classes, etc.
  - Give names starting with the underscore to class members that are not public by their intention. Don't use these members outside of the class which they belong to.
  - Docstrings for all public modules and functions/methods. Pay special attention to describing argument list
  - Annotate types for arguments of public functions

  - Annotate return type for public functions
  - Use flake8 in developer environment and CI (fail a build if the flake8 check does not pass). (you can run `flake8 .` on the project root to check it; you can install flake8 from pypi: `pip install flake8`)
- Inheritance and polymorphism
  - Use ABC when creating abstract class to ensure that all its fields implemented in successors.
  - Prefer composition instead of multiple inheritance
  - Avoid creating deep hierarchies
  - Avoid type checks (type(instance) == specific_subclass), we have them in some places and it is awful, do polymorphism instead
  - If there is an interface or abstract class then it should be used everywhere instead of specific instance
  - Make sure that methods of subclass are fully compatible with their declarations in interface/abstract class
- Separation of concerns
  - Follow separation of concerns principles
  - Make components as independent as possible
- Avoid high coupling
  - Some classes have references to each other, for example Node and Replica, so we get a kind of "spaghetti code
- Use classes instead of parallel arrays
- Clear and not duplicated Utilities
- Decomposition
  - There are functions and classes which are too long. For example class Node has more than 2000 lines of code, this complicates understanding of logic.
  - This can be solved by destructuring of such classes or functions on a smaller one by aggregation, composition and inheritance
- No multiple enclosed if-elif-else statements
  - It's hard to read the code containing multiple enclosed if-else statements
  - Consider checking some conditions a the beginning of a method and return immediately
- Choose good variable and class names
  - Make variable and name classes to be clear, especially for Public API
  - Avoid unclear abbreviations in public API
- Use asyncio instead of callbacks
  - It looks and behaves just like a method call - you know that further code is not executed until coroutine completed.
  - It requires no callback preparation - code is cleaner
  - It does not break context - exception raised in a coroutine has the same stack trace as an exception raised in a method (while the one raised in callback don't)
  - It is easy to chain coroutine calls
  - If it is needed you can easily wrap coroutine in a Future and execute it in non-blocking fashion

- Consider using Actor models (this is our long-term goal for re-factoring to achieve better code quality and performance)
- Think about performance of the pool

12.3 Test quality items

- Write good tests
  - Not all of our tests are clean and clear enough
  - Follow TDD in writing the tests first
  - Have Unit tests where possible/necessary
  - Avoid using 'module' level fixtures if possible
- Use indy-sdk in all tests where possible. If it's not possible to use indy-sdk for some reasons (for example, indy-sdk doesn't have a required feature), then log a ticket in Indy-sdk Jira and put a TODO comment in the test.
- Use `txnPoolNodeSet`, not `nodeSet` fixture. `nodeSet` will be deprecated soon.
- Importing fixtures
  - Fixture imports are not highlighted by IDEs, that's why they can be accidently removed by someone. To avoid this try to move fixture to conftest of the containing package or conftest of some of higher-level package - this allows fixture to be resolved automatically. If you still import fixtures, mark such the imports with the following hint for IDE not to mark them as unused and not to remove them when optimizing imports:

    ```
    # noinspection PyUnresolvedReferences
    ```
- Try to use the same config and file folder structure for integration tests as in real environment. As of now, all tests follow the same file folder structure (see *indy-file-structure-guideline*) as Indy-node service, but the folders are created inside `tmp` test folder.

CHAPTER 13

Dev Setup

There are scripts that can help in setting up environment and project for developers. The scripts are in dev-setup folder.

Note: as of now, we provide scripts for Ubuntu only. It's not guaranteed that the code is working on Windows.

- One needs Python 3.5 to work with the code
- We recommend using Python virtual environment for development
- We use pytest for unit and integration testing
- There are some dependencies that must be installed before being able to run the code

13.1 Quick Setup on Ubuntu 16.04:

This is a Quick Setup for development on a clean Ubuntu 16.04 machine. You can also have a look at the scripts mentioned below to follow them and perform setup manually.

1. Get scripts from dev-setup-ubuntu
2. Run `setup-dev-python.sh` to setup Python3.5, pip and virtualenv
3. Run `source ~/.bashrc` to apply virtual environment wrapper installation
4. Run `setup-dev-depend-ubuntu16.sh` to setup dependencies (libindy, libindy-crypto, libsodium)
5. Fork indy-plenum and indy-node
6. Go to the destination folder for the project
7. Run `init-dev-project.sh <github-name> <new-virtualenv-name>` to clone indy-plenum and indy-node projects and create a virtualenv to work in
8. Activate new virtualenv `workon <new-virtualenv-name>`
9. [Optionally] Install Pycharm
10. [Optionally] Open and configure projects in Pycharm:

- Open both indy-plenum and indy-node in one window
- Go to `File -> Settings`
- Configure Project Interpreter to use just created virtualenv
  - Go to `Project: <name> -> Project Interpreter`
  - You'll see indy-plenum and indy-node projects on the right side tab. For each of them:
    * Click on the project just beside "Project Interpreter" drop down, you'll see one setting icon, click on it.
    * Select "Add Local"
    * Select existing virtualenv path as below: /bin/python3.5 For example: /home/user_name/.virtualenvs/new-virtualenv-name>/bin/python3.5
- Configure Project Dependency
  - Go to `Project: <name> -> Project Dependencies`
  - Mark each project to be dependent on another one
- Configure pytest
  - Go to `Configure Tools -> Python Integrated tools`
  - You'll see indy-plenum and indy-node projects on the right side tab. For each of them:
    * Select Py.test from the 'Default test runner'
- Press `Apply`

13.2 Detailed Setup

13.2.1 Setup Python

One needs Python 3.5 to work with the code. You can use `dev-setup/ubuntu/setup_dev_python.sh` script for quick installation of Python 3.5, pip and virtual environment on Ubuntu, or follow the detailed instructions below.

Ubuntu

1. Run `sudo add-apt-repository ppa:deadsnakes/ppa`
2. Run `sudo apt-get update`
3. On Ubuntu 14, run `sudo apt-get install python3.5` (python3.5 is pre-installed on most Ubuntu 16 systems; if not, do it there as well.)

CentOS/Redhat

Run `sudo yum install python3.5`

Mac

1. Go to python.org and from the "Downloads" menu, download the Python 3.5.0 package (python-3.5.0-macosx10.6.pkg) or later.
2. Open the downloaded file to install it.
3. If you are a homebrew fan, you can install it using this brew command: `brew install python3`
4. To install homebrew package manager, see: brew.sh

Windows

Download the latest build (pywin32-220.win-amd64-py3.5.exe is the latest build as of this writing) from here and run the downloaded executable.

13.2.2 Setup Libsodium

Indy also depends on libsodium, an awesome crypto library. These need to be installed separately.

Ubuntu

1. We need to install libsodium with the package manager. This typically requires a package repo that's not active by default. Inspect `/etc/apt/sources.list` file with your favorite editor (using sudo). On ubuntu 16, you are looking for a line that says `deb http://us.archive.ubuntu.com/ubuntu xenial main universe`. On ubuntu 14, look for or add: `deb http://ppa.launchpad.net/chris-lea/libsodium/ubuntu trusty main` and `deb-src http://ppa.launchpad.net/chris-lea/libsodium/ubuntu trusty main`.
2. Run `sudo apt-get update`. On ubuntu 14, if you get a GPG error about public key not available, run this command and then, after, retry apt-get update: `sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys B9316A7BC7917B12`
3. Install libsodium; the version depends on your distro version. On Ubuntu 14, run `sudo apt-get install libsodium13`; on Ubuntu 16, run `sudo apt-get install libsodium18`

CentOS/Redhat

Run `sudo yum install libsodium-devel`

Mac

Once you have homebrew installed, run `brew install libsodium` to install libsodium.

Windows

1. Go to https://download.libsodium.org/libsodium/releases/ and download the latest libsodium package (libsodium-1.0.8-mingw.tar.gz is the latest version as of this writing).
2. When you extract the contents of the downloaded tar file, you will see 2 folders with the names libsodium-win32 and libsodium-win64.

3. As the name suggests, use the libsodium-win32 if you are using 32-bit machine or libsodium-win64 if you are using a 64-bit operating system.
4. Copy the libsodium-x.dll from libsodium-win32\bin or libsodium-win64\bin to C:\Windows\System or System32 and rename it to libsodium.dll.

13.2.3 Setup Indy-Crypto

Indy depends on Indy-Crypto.

There is a deb package of libindy-crypto that can be used on Ubuntu:

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys CE7709D068DB5E88
sudo add-apt-repository "deb https://repo.sovrin.org/deb xenial master"
sudo apt-get update
sudo apt-get install libindy-crypto
```

See Indy-Crypto on how it can be installed on other platforms.

13.2.4 Setup RocksDB

Indy depends on RocksDB, an embeddable persistent key-value store for fast storage.

Currently Indy requires RocksDB version 5.8.8 or higher. There is a deb package of RocksDB-5.8.8 and related stuff that can be used on Ubuntu 16.04 (repository configuration steps may be skipped if Indy-Crypto installation steps have been done):

```
# Start of repository configuration steps
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys CE7709D068DB5E88
sudo add-apt-repository "deb https://repo.sovrin.org/deb xenial master"
# End of repository configuration steps
sudo apt-get update
sudo apt-get install libbz2-dev \
    zlib1g-dev \
    liblz4-dev \
    libsnappy-dev \
    rocksdb=5.8.8
```

See RocksDB on how it can be installed on other platforms.

13.2.5 Setup Libindy

Indy needs Libindy as a test dependency.

There is a deb package of libindy that can be used on Ubuntu:

```
sudo add-apt-repository "deb https://repo.sovrin.org/sdk/deb xenial stable"
sudo apt-get update
sudo apt-get install -y libindy
```

See Libindy on how it can be installed on other platforms.

13.2.6 Using a virtual environment (recommended)

We recommend creating a new Python virtual environment for trying out Indy. A virtual environment is a Python environment which is isolated from the system's default Python environment (you can change that) and any other virtual environment you create.

You can create a new virtual environment by:

```
virtualenv -p python3.5 <name of virtual environment>
```

And activate it by:

```
source <name of virtual environment>/bin/activate
```

Optionally, you can install virtual environment wrapper as follows:

```
pip3 install virtualenvwrapper
echo '' >> ~/.bashrc
echo '# Python virtual environment wrapper' >> ~/.bashrc
echo 'export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3' >> ~/.bashrc
echo 'export WORKON_HOME=$HOME/.virtualenvs' >> ~/.bashrc
echo 'source /usr/local/bin/virtualenvwrapper.sh' >> ~/.bashrc
source ~/.bashrc
```

It allows to create a new virtual environment and activate it by using

```
mkvirtualenv -p python3.5 <env_name>
workon <env_name>
```

13.2.7 Installing code and running tests

Activate the virtual environment.

Navigate to the root directory of the source (for each project) and install required packages by

```
pip install -e .[tests]
```

If you are working with both indy-plenum and indy-node, then please make sure that both projects are installed with -e option, and not from pypi (have a look at the sequence at `init-dev-project.sh`).

Go to the folder with tests (either `indy-plenum`, `indy-node/indy_node`, `indy-node/indy_client` or `indy-node/indy_common`) and run tests

```
pytest .
```