Exonum Python light client

Release 0.3.0

The Exonum team

Jan 21, 2020

CONTENTS:

CHAPTER

ONE

OVERVIEW

Exonum Light Client is a Python library for working with Exonum blockchain from the client side. It can be easily integrated to an existing application. Also, Exonum Light Client provides access to common utils toolkit which contains some helpful functions for hashing, cryptography, serialization, etc.

CHAPTER

TWO

CAPABILITIES

By using the client you are able to perform the following operations:

- Submit transactions to the node
- Receive information on transactions
- Receive information on blockchain blocks
- Receive information on the node system
- Receive information on the node status

CHAPTER

THREE

SYSTEM DEPENDENCIES

- Python 3.5 or above.
- Package installer for Python3 (pip3)

CHAPTER

FOUR

EXAMPLES

The following example shows how to create an instance of the Exonum client which will be able to work with an Exonum node with the Cryptocurrency Advanced service mounted on it, at http://localhost:8080 address:

4.1 Installing Python Light Client

First of all, we need to install our client library:

```
git clone git@github.com:exonum/exonum-python-client.git
pip3 install -e exonum-python-client
```

4.2 Exonum Client Initialization

```
>>> from exonum_client import ExonumClient, ModuleManager, MessageGenerator
>>> from exonum_client.crypto import KeyPair
>>> client = ExonumClient(hostname="localhost", public_api_port=8080, private_api_
↪port=8081, ssl=False)
```

4.3 Compiling Proto Files

To compile proto files into the Python analogues we need a Protobuf loader:

```
>>> with client.protobuf_loader() as loader:
>>>     #  Your code goes here.
```

Since loader acquires resources on initialization, it is recommended that you create the loader via the context manager. Otherwise you should initialize and deinitialize the client manually:

```
>>> loader = client.protobuf_loader()
>>> loader.initialize()
>>> # ... Some usage
>>> loader.deinitialize()
```

Then we need to run the following code:

```
>>> loader.load_main_proto_files()  # Loads and compiles main proto files, such as␣
↪`runtime.proto`, `consensus.proto`, etc.
>>> loader.load_service_proto_files(runtime_id=0, service_name='exonum-supervisor:0.
↪12.0')  # Same for a specific service.
```

- runtime_id=0 here means, that service works in Rust runtime.

4.4 Creating Transaction Messages

The following example shows how to create a transaction message:

```
>>> alice_keys = KeyPair.generate()
>>>
>>> cryptocurrency_service_name = 'exonum-cryptocurrency-advanced:0.12.0'
>>> loader.load_service_proto_files(runtime_id=0, service_name=cryptocurrency_service_
↪name)
>>>
>>> cryptocurrency_module = ModuleManager.import_service_module(cryptocurrency_
↪service_name, 'service')
>>> cryptocurrency_message_generator = MessageGenerator(service_id=1024, artifact_
↪name=cryptocurrency_service_name)
>>>
>>> create_wallet_alice = cryptocurrency_module.CreateWallet()
>>> create_wallet_alice.name = 'Alice'
>>> create_wallet_alice_tx = cryptocurrency_message_generator.create_message(create_
↪wallet_alice)
>>> create_wallet_alice_tx.sign(alice_keys)
```

- 1024 - service instance ID.
- alice_keys - public and private keys of the ed25519 public-key signature system.

After invoking the sign method we get a signed transaction. This transaction is ready for sending to an Exonum node.

4.5 Sending Transaction to the Exonum Node

After successfully sending the message, we'll get a response which will contain a hash of the transaction:

```
>>> response = client.public_api.send_transaction(create_wallet_alice_tx)
{
    "tx_hash": "3541201bb7f367b802d089d8765cc7de3b7dfc253b12330b8974268572c54c01"
}
```

4.6 Subscribing to events

If you want to subscribe to events (subscription_type: "transactions" or "blocks"), use the following code:

```
>>> with client.create_subscriber(subscription_type="blocks") as subscriber:
>>>     subscriber.wait_for_new_event()
>>>     subscriber.wait_for_new_event()
```

Context manager will automatically create a connection and will disconnect after use. Or you can manually do the same:

```
>>> subscriber = client.create_subscriber(subscription_type="blocks")
>>> subscriber.connect()
>>> # ... Your code
>>> subscriber.stop()
```

Keep in mind that if you forget to stop the subscriber, you may discover HTTP errors when you try to use Exonum API.

4.7 Getting Data on the Available Services

The code below will show a list of the artifacts available for the start and a list of working services:

```
>>> client.public_api.available_services().json()
{
  'artifacts': [
    {
      'runtime_id': 0,
      'name': 'exonum-cryptocurrency-advanced',
      'version': '0.12.0'
    },
    {
      'runtime_id': 0,
      'name': 'exonum-supervisor',
      'version': '0.12.0'
    }
  ],
  'services': [
    {
      'spec': {
        'id': 1024,
        'name': 'XNM',
        'artifact': {
          'runtime_id': 0,
          'name': 'exonum-cryptocurrency-advanced',
          'version': '0.12.0'
        }
      },
      'status': 'Active'
    },
    {
      'spec': {
        'id': 0,
        'name': 'supervisor',
        'artifact': {
          'runtime_id': 0,
          'name': 'exonum-supervisor',
          'version': '0.12.0'
        }
      },
      'status': 'Active'
    }
  ]
}
```

4.8 More Examples

You can find the sample scripts in the GitHub repository examples section:

CHAPTER

FIVE

MODULES DOCUMENTATION

Documentation for the modules in the Exonum Python Light Client:

5.1 exonum

5.1.1 exonum package

Module Contents

Subpackages

exonum_client.proofs package

Module Contents

Subpackages

exonum_client.proofs.list_proof package

Module Contents

Submodules

exonum_client.proofs.list_proof.list_proof module

exonum_client.proofs.list_proof.errors module

exonum_client.proofs.map_proof package

Module Contents

Submodules

exonum_client.proofs.map_proof.map_proof_builder module

exonum_client.proofs.map_proof.map_proof module

exonum_client.proofs.map_proof.errors module

Submodules

exonum_client.proofs.encoder module

Submodules

exonum_client.client module

exonum_client.crypto module

exonum_client.message module

exonum_client.module_manager module

exonum_client.protobuf_loader module

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search