Exonum Python light client

Release 0.3.0

Jan 21, 2020

Contents:

CHAPTER 1

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 2

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 3

Compatibility

The following table shows versions compatibility:

Light Client	Exonum
0.1	0.9.*
0.2	0.10.*
0.3	0.12.1+
master	dynamic_services branch

CHAPTER 4

System Dependencies

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

CHAPTER 5

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:

5.1 Installing Python Light Client

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

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

5.2 Exonum Client Initialization

```
>>> from exonum import ExonumClient, MessageGenerator, ModuleManager, gen_keypair
>>> client = ExonumClient(hostname="localhost", public_api_port=8080, private_api_
↪port=8081, ssl=False)
```

5.3 Compiling Proto Files

To compile proto files into the Python analogues we need a Protobuf Provider and a Protobuf Loader.

Protobuf provider objects accept either file system paths or github public pages.

```
>>> from exonum.protobuf_provider import ProtobufProvider
>>> main_sources_url = "https://github.com/exonum/exonum/tree/v0.12/exonum/src/proto/
↪schema/exonum"
>>> cryptocurrency_sources_url = (
>>>     "https://github.com/exonum/exonum/tree/v0.12/examples/cryptocurrency-advanced/
↪backend/src/proto"
```

(continues on next page)

(continued from previous page)

```
>>> )
>>> protobuf_provider = ProtobufProvider()
>>> protobuf_provider.add_source(main_sources_url)
>>> protobuf_provider.add_source(cryptocurrency_sources_url, "cryptocurrency-advanced
↪")
```

After creating a protobuf provider, you need to set it for the client.

```
>>> client.set_protobuf_provider(protobuf_provider)
```

Now you're ready to use 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(0, service_name='cryptocurrency-advanced')  #␣
↪Same for specific service.
```

- first argument for *load_service_proto_files* should always be 0.

5.4 Creating Transaction Messages

The following example shows how to create a transaction message:

```
>>> from exonum.crypto import KeyPair
>>> keys = KeyPair.generate()
>>>
>>> cryptocurrency_service_name = "cryptocurrency-advanced"
>>> loader.load_service_proto_files(runtime_id=0, cryptocurrency_service_name)
>>>
>>> cryptocurrency_module = ModuleManager.import_service_module(cryptocurrency_
↪service_name, "cryptocurrency")
>>> cryptocurrency_message_generator = MessageGenerator(128, cryptocurrency_service_
↪name, "cryptocurrency")
>>>
>>> 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(keys)
```

- 128 - service ID.

- key_pair - public and private keys of the ed25519 public-key signature system.
- “cryptocurrency” means “cryptocurrency.proto” file.

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

5.5 More Examples

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

CHAPTER 6

Modules Documentation

Documentation for the modules in the Exonum Python Light Client:

6.1 exonum

6.1.1 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

CHAPTER 7

Indices and Tables

- genindex
- modindex
- search