Pymaker

A Python API for the Maker Smart Contracts

Introduction

The Maker Protocol incentivizes external agents, calledkeepers , to automate certain operations around the Ethereum blockchain. In order to ease their development, an API around most of the Maker contracts has been created. It can be used not only by keepers, but may also be found useful by authors of some other, unrelated utilities aiming to interact with these contracts.

Based on thePymaker API , a set of reference Maker keepers is being developed. They all used to reside in this repository, but now each of them has an individual one:bite-keeper (SCD only),arbitrage-keeper ,auction-keeper (MCD only),cdp-keeper (SCD only),market-maker-keeper .

You only need to install this project directly if you want to build your own keepers, or if you want to play with this API library itself. If you just want to install one of reference keepers, go to one of the repositories linked above and start from there. Each of these keepers references some version ofpymaker via a Git submodule.

Installation

This project usesPython 3.6.6 .

In order to clone the project and install required third-party packages please execute:

```
Copy git clone https://github.com/makerdao/pymaker.git cd pymaker pip3 install -r requirements.txt
```

Known Ubuntu issues

In order for thesecp256k Python dependency to compile properly, following packages will need to be installed:

```
Copy sudo apt-get install build-essential automake libtool pkg-config libffi-dev python-dev python-pip libsecp256k1-dev
```

(for Ubuntu 18.04 Server)

Known macOS issues

In order for the Python requirements to install correctly onmacOS , please installopenssl ,libtool ,pkg-config andautomake usingHomebrew :

```
Copy brew install openssl libtool pkg-config automake
```

and set theLDFLAGS environment variable before you runpip3 install -r requirements.txt :

```
Copy export LDFLAGS="-L(brew --prefix openssl)/lib" CFLAGS="-I(brew --prefix openssl)/include"
```

Available APIs

The current version provides APIs around:

- ERC20Token
- ,
- Tub
- ,Tap
- ,Top

- andVox
- (https://github.com/makerdao/sai
-),
- Vat
- ,Cat
- ,Vow
- ,Jug
- ,Flipper
- ,Flapper
- ,Flopper
- (https://github.com/makerdao/dss
-)
- SimpleMarket
- ,ExpiringMarket
- andMatchingMarket
- (https://github.com/makerdao/maker-otc
-),
- TxManager
- (https://github.com/makerdao/tx-manager
-),
- DSGuard
- (https://github.com/dapphub/ds-guard
-),
- DSToken
- (https://github.com/dapphub/ds-token
-),
- DSEthToken
- (https://github.com/dapphub/ds-eth-token
-),
- DSValue
- (https://github.com/dapphub/ds-value
-),
- DSVault
- (https://github.com/dapphub/ds-vault
-),
- EtherDelta
- (https://github.com/etherdelta/etherdelta.github.io
-),
- 0x v1
- (https://etherscan.io/address/0x12459c951127e0c374ff9105dda097662a027093#code
- ,https://github.com/0xProject/standard-relayer-api
-),
- 0x v2
- Dai Savings Rate (Pot)
- (https://github.com/makerdao/pymaker/blob/master/tests/manual_test_dsr.py#L29
-)
-

APIs around the following functionality have not been implemented:

- Global Settlement (End)
- Governance (DSAuth, DSChief, DSGuard, DSSpell, Mom)
-

Contributions from the community are much appreciated!

Code samples

Below you can find some code snippets demonstrating how the API can be used both for developing your own keepers and for creating some other utilities interacting with the Maker Protocol ecosystem contracts.

Token transfer

This snippet demonstrates how to transfer some SAI from our default address. The SAI token address is discovered by querying theTub , so all we need as aTub address:

```

Copy from web3 import HTTPProvider, Web3
```

from pymaker import Address from pymaker.token import ERC20Token from pymaker.numeric import Wad from pymaker.sai import Tub

web3 = Web3(HTTPProvider(endpoint_uri="http://localhost:8545"))

tub = Tub(web3=web3, address=Address(' 0xb7ae5ccabd002b5eebafe6a8fad5499394f67980')) sai = ERC20Token(web3=web3, address=tub.sai())

sai.transfer(address=Address(' 0x0000000000111111111100000000001111111111'), value=Wad.from_number(10)).transact()

```

Updating a DSValue

This snippet demonstrates how to update aDSValue with the ETH/USD rate pulled fromCryptoCompare :

```

Copy import json import urllib.request

from web3 import HTTPProvider, Web3

from pymaker import Address from pymaker.feed import DSValue from pymaker.numeric import Wad

def cryptocompare_rate() -> Wad: with urllib.request.urlopen("https://min-api.cryptocompare.com/data/price?fsym=ETH&tsyms=USD") as url: data = json.loads(url.read().decode()) return Wad.from_number(data['USD'])

web3 = Web3(HTTPProvider(endpoint_uri="http://localhost:8545"))

dsvalue = DSValue(web3=web3, address=Address(' 0x038b3d8288df582d57db9be2106a27be796b0daf')) dsvalue.poke_with_int(cryptocompare_rate().value).transact()

```

SAI introspection

This snippet demonstrates how to fetch data fromTub andTap contracts:

```

Copy from web3 import HTTPProvider, Web3

from pymaker import Address from pymaker.token import ERC20Token from pymaker.numeric import Ray from pymaker.sai import Tub, Tap

web3 = Web3(HTTPProvider(endpoint_uri="http://localhost:8545"))

tub = Tub(web3=web3, address=Address(' 0x448a5065aebb8e423f0896e6c5d525c040f59af3')) tap = Tap(web3=web3, address=Address(' 0xbda109309f9fafa6dd6a9cb9f1df4085b27ee8ef')) sai = ERC20Token(web3=web3, address=tub.sai()) skr = ERC20Token(web3=web3, address=tub.skr()) gem = ERC20Token(web3=web3, address=tub.gem())

print(f"") print(f"Token summary") print(f"-------------") print(f"SAI total supply : {sai.total_supply()} SAI") print(f"SKR total supply : {skr.total_supply()} SKR") print(f"GEM total supply : {gem.total_supply()} GEM") print(f"") print(f"Collateral summary") print(f"------------------") print(f"GEM collateral : {tub.pie()} GEM") print(f"SKR collateral : {tub.air()} SKR") print(f"SKR pending liquidation: {tap.fog()} SKR") print(f"") print(f"Debt summary") print(f"------------") print(f"Debt ceiling : {tub.cap()} SAI") print(f"Good debt : {tub.din()} SAI") print(f"Bad debt : {tap.woe()} SAI") print(f"Surplus : {tap.joy()} SAI") print(f"") print(f"Feed summary") print(f"------------") print(f"REF per GEM feed : {tub.pip()}") print(f"REF per SKR price : {tub.tag()}") print(f"GEM per SKR price : {tub.per()}") print(f"") print(f"Tub parameters") print(f"--------------") print(f"Liquidation ratio : {tub.mat()*100} %") print(f"Liquidation penalty : {tub.axe()*100 - Ray.from_number(100)} %") print(f"Stability fee : {tub.tax()} %") print(f"") print(f"All cups") print(f"--------") for cup_id in range(1, tub.cupi()+1): cup = tub.cups(cup_id) print(f"Cup #{cup_id}, lad={cup.lad}, ink={cup.ink} SKR, tab={tub.tab(cup_id)} SAI, safe={tub.safe(cup_id)}")

```

Multi-collateral Dai

This snippet demonstrates how to create a CDP and draw Dai.

```

Copy import sys from web3 import Web3, HTTPProvider from pymaker import Address from pymaker.deployment import DssDeployment from pymaker.keys import register_keys from pymaker.numeric import Wad

web3 = Web3(HTTPProvider(endpoint_uri="https://localhost:8545", request_kwargs={"timeout": 10})) web3.eth.defaultAccount = sys.argv[1] # ex: 0x0000000000000000000000000000000aBcdef123 register_keys(web3, [sys.argv[2]]) # ex: key_file=~keys/default-account.json,pass_file=~keys/default-account.pass

mcd = DssDeployment.from_json(web3=web3, conf=open("tests/config/kovan-addresses.json", "r").read()) our_address = Address(web3.eth.defaultAccount)

Choose the desired collateral; in this case we'll wrap some Eth

collateral = mcd.collaterals['ETH-A'] ilk = collateral.ilk collateral.gem.deposit(Wad.from_number(3)).transact()

Add collateral and allocate the desired amount of Dai

collateral.approve(our_address) collateral.adapter.join(our_address, Wad.from_number(3)).transact() mcd.vat.frob(ilk, our_address, dink=Wad.from_number(3), dart=Wad.from_number(153)).transact() print(f"CDP Dai balance before withdrawal: {mcd.vat.dai(our_address)}")

Mint and withdraw our Dai

mcd.approve_dai(our_address) mcd.dai_adapter.exit(our_address, Wad.from_number(153)).transact() print(f"CDP Dai balance after withdrawal: {mcd.vat.dai(our_address)}")

Repay (and burn) our Dai

assert mcd.dai_adapter.join(our_address, Wad.from_number(153)).transact() print(f"CDP Dai balance after repayment: {mcd.vat.dai(our_address)}")

Withdraw our collateral

mcd.vat.frob(ilk, our_address, dink=Wad(0), dart=Wad.from_number(-153)).transact() mcd.vat.frob(ilk, our_address, dink=Wad.from_number(-3), dart=Wad(0)).transact() collateral.adapter.exit(our_address, Wad.from_number(3)).transact() print(f"CDP Dai balance w/o collateral: {mcd.vat.dai(our_address)}")

```

Asynchronous invocation of Ethereum transactions

This snippet demonstrates how multiple token transfers can be executed asynchronously:

```

Copy from web3 import HTTPProvider from web3 import Web3

from pymaker import Address, synchronize from pymaker.numeric import Wad from pymaker.sai import Tub from pymaker.token import ERC20Token

web3 = Web3(HTTPProvider(endpoint_uri="http://localhost:8545"))

tub = Tub(web3=web3, address=Address(' 0x448a5065aebb8e423f0896e6c5d525c040f59af3')) sai = ERC20Token(web3=web3, address=tub.sai()) skr = ERC20Token(web3=web3, address=tub.skr())

synchronize([sai.transfer(Address(' 0x0101010101020202020203030303030404040404'), Wad.from_number(1.5)).transact_async(), skr.transfer(Address(' 0x0303030303040404040405050505050606060606'), Wad.from_number(2.5)).transact_async()])

```

Multiple invocations in one Ethereum transaction
```

This snippet demonstrates how multiple token transfers can be executed in one Ethereum transaction. ATxManager instance has to be deployed and owned by the caller.

```

Copy from web3 import HTTPProvider from web3 import Web3

from pymaker import Address from pymaker.approval import directly from pymaker.numeric import Wad from pymaker.sai import Tub from pymaker.token import ERC20Token from pymaker.transactional import TxManager

web3 = Web3(HTTPProvider(endpoint_uri="http://localhost:8545"))

tub = Tub(web3=web3, address=Address(' 0x448a5065aebb8e423f0896e6c5d525c040f59af3')) sai = ERC20Token(web3=web3, address=tub.sai()) skr = ERC20Token(web3=web3, address=tub.skr())

tx = TxManager(web3=web3, address=Address(' 0x57bFE16ae8fcDbD46eDa9786B2eC1067cd7A8f48')) tx.approve([sai, skr], directly())

tx.execute([sai.address, skr.address], [sai.transfer(Address(' 0x0101010101020202020203030303030404040404'), Wad.from_number(1.5)).invocation(), skr.transfer(Address(' 0x0303030303040404040405050505050606060606'), Wad.from_number(2.5)).invocation()]).transact()

```

Ad-hoc increasing of gas price for asynchronous transactions

```

Copy import asyncio from random import randint

from web3 import Web3, HTTPProvider

from pymaker import Address from pymaker.gas import FixedGasPrice from pymaker.oasis import SimpleMarket

web3 = Web3(HTTPProvider(endpoint_uri=f"http://localhost:8545")) otc = SimpleMarket(web3=web3, address=Address(' 0x375d52588c3f39ee7710290237a95C691d8432E7'))

async def bump_with_increasing_gas_price(order_id): gas_price = FixedGasPrice(gas_price=1000000000) task = asyncio.ensure_future(otc.bump(order_id).transact_async(gas_price=gas_price))

while not task.done(): await asyncio.sleep(1) gas_price.update_gas_price(gas_price.gas_price + randint(0, gas_price.gas_price))

return task.result()

bump_task = asyncio.ensure_future(bump_with_increasing_gas_price(otc.get_orders()[-1].order_id)) event_loop = asyncio.get_event_loop() bump_result = event_loop.run_until_complete(bump_task)

print(bump_result) print(bump_result.transaction_hash)

```

Testing

Prerequisites:

- docker and docker-compose
- ganache-cli
- 6.2.5
- (using npm,sudo npm install -g ganache-cli@6.2.5
-)
-

This project usespytest for unit testing. Testing of Multi-collateral Dai is performed on a Dockerized local testchain included intests\config .

In order to be able to run tests, please install development dependencies first by executing:

```

Copy pip3 install -r requirements-dev.txt
```

```

You can then run all tests with:

```
Copy ./test.sh
```

If you have questions regarding Pymaker, please reach out to us on the#keeper channel onchat.makerdao.com .

Previous Exchange Service Next The Auctions of the Maker Protocol Last updated4 years ago On this page *Introduction * Installation * Available APIs * Code samples * Testing

Export as PDF
```