aiosend

Release 3.0.6

VoVcHiC

Jun 16, 2026

CONTENTS

1 Features 3

2 Quick start 5

3 Contents 7

Python Module Index 51

Index 53

aiosend is a synchronous & asynchronous Crypto Pay API client.

See also

aiosend has community chat on Telegram

CHAPTER

ONE

FEATURES

- supports both synchronous and asynchronous use
- provides *invoice polling* and *check polling*
- provides *webhook handling*
- fully typed, has *literal type hints and warnings*
- uses powerful magic filters from aiogram 3.x
- provides *dependency injection*
- provides *additional tool methods*
- provides *shortcut methods for types*

CHAPTER

TWO

QUICK START

```python
import asyncio
from aiosend import CryptoPay

async def main() -> None:
    cp = CryptoPay(token="TOKEN")
    app = await cp.get_me()
    print(app.name)  # Your App's Name

if __name__ == "__main__":
    asyncio.run(main())
```

CHAPTER

THREE

CONTENTS

3.1 Installation

3.1.1 via pip

- from PyPI

```
pip install -U aiosend
```

- from GitHub

```
pip install https://github.com/vovchic17/aiosend/archive/refs/heads/main.zip
```

3.1.2 via uv

```
uv add aiosend
```

3.1.3 via Poetry

```
poetry add aiosend
```

3.1.4 via PDM

```
pdm add aiosend
```

3.1.5 via pipx

```
pipx install aiosend
```

3.1.6 via Rye

```
rye add aiosend
```

3.1.7 via Conda

```
conda install aiosend
```

3.1.8 via Pipenv

```
pipenv install aiosend
```

3.2 API

- **Methods** - official Crypto Pay API methods.
- **Types** - official Crypto Pay API types.
- **Enums** - enumerable parameter values of methods.
- **Network** - API networks.
- **Errors** - error models.

3.2.1 Methods

Crypto Pay API methods implementation.

`CryptoPay.get_me()`

getMe method.

Use this method to test your app's authentication token. Requires no parameters. On success, returns basic information about an app.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_f4a0fd9ca0

Return type
App

`CryptoPay.create_invoice(`*amount*, *asset=None*, ** (Keyword-only parameters separator (PEP 3102))*, *currency_type=None*, *fiat=None*, *accepted_assets=None*, *swap_to=None*, *description=None*, *hidden_message=None*, *paid_btn_name=None*, *paid_btn_url=None*, *payload=None*, *allow_comments=None*, *allow_anonymous=None*, *expires_in=None*`)`

createInvoice method.

Use this method to create a new invoice. On success, returns an object of the created `aiosend.types.Invoice`.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_38468af1a6

Parameters

- **amount** (float) – Amount of the invoice in float. For example: *125.50*
- **currency_type** (*CurrencyType* | None) – *Optional*. Type of the price, can be "crypto" or "fiat". Defaults to crypto.
- **asset** (*Asset* | None) – *Optional*. Required if currency_type is "crypto". Cryptocurrency alphabetic code. Supported assets: "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC".
- **fiat** (*Fiat* | None) – *Optional*. Required if currency_type is "fiat". Fiat currency code. Supported fiat currencies: "USD", "EUR", "RUB", "BYN", "UAH", "GBP", "CNY", "KZT", "UZS", "GEL", "TRY", "AMD", "THB", "INR", "BRL", "IDR", "AZN", "AED", "PLN" and "ILS".
- **accepted_assets** (list[*Asset*] | None) – *Optional*. List of cryptocurrency alphabetic codes. Assets which can be used to pay the invoice. Available only if currency_type is

"fiat". Supported assets: "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet). Defaults to all currencies.

- **swap_to** (*Asset* | None) – *Optional*. The asset that will be attempted to be swapped into after the user makes a payment (the swap is not guaranteed). Supported assets: "USDT", "TON", "TRX", "ETH", "SOL", "BTC", "LTC".
- **description** (str | None) – *Optional*. Description for the invoice. User will see this description when they pay the invoice. Up to 1024 characters.
- **hidden_message** (str | None) – *Optional*. Text of the message which will be presented to a user after the invoice is paid. Up to 2048 characters.
- **paid_btn_name** (*PaidBtnName* | None) – *Optional*. Label of the button which will be presented to a user after the invoice is paid.
- **paid_btn_url** (str | None) – *Optional*. Required if paid_btn_name is specified. URL opened using the button which will be presented to a user after the invoice is paid. You can set any callback link (for example, a success link or link to homepage). Starts with https or http.
- **payload** (str | None) – *Optional*. Any data you want to attach to the invoice (for example, user ID, payment ID, ect). Up to 4kb.
- **allow_comments** (bool | None) – *Optional*. Allow a user to add a comment to the payment. Defaults to True.
- **allow_anonymous** (bool | None) – *Optional*. Allow a user to pay the invoice anonymously. Defaults to True.
- **expires_in** (int | None) – *Optional*. You can set a payment time limit for the invoice in seconds. Values between 1-2678400 are accepted.

Return type
Invoice

CryptoPay.**delete_invoice**(*invoice_id*)

deleteInvoice method.

Use this method to delete invoices created by your app. Returns True on success.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_da2ea9f39c

Parameters
invoice_id (int) – Invoice ID to be deleted.

Return type
bool

Tip

To use /create_check method you need to enable it to the restriction settings in @CryptoBot as follows:

Crypto Pay -> My Apps -> <Your App> -> Security -> Checks -> Enable.

CryptoPay.**create_check**(*amount*, *asset*, *pin_to_user_id=None*, *pin_to_username=None*)

createCheck method.

Use this method to create a new check. On success, returns an object of the created *aiosend.types.Check*.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_cef427b32c

Parameters

- **amount** (float) – Amount of the check in float. For example: *125.50*
- **asset** (*Asset*) – Cryptocurrency alphabetic code. Supported assets: "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet).
- **pin_to_user_id** (int | None) – *Optional*. ID of the user who will be able to activate the check.
- **pin_to_username** (str | None) – *Optional*. A user with the specified username will be able to activate the check.

Return type

Check

CryptoPay.**delete_check**(*check_id*)

deleteCheck method.

Use this method to delete checks created by your app. Returns True on success.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_b27428d56a

Parameters

check_id (int) – Check ID to be deleted.

Return type

bool

> **Tip**
>
> To use /transfer method you need to enable it to the restriction settings in @CryptoBot as follows:
>
> Crypto Pay -> My Apps -> <Your App> -> Security -> Transfers -> Enable.

CryptoPay.**transfer**(*user_id*, *asset*, *amount*, *spend_id=None*, *comment=None*, ***, *disable_send_notification=None*)

transfer method.

Use this method to send coins from your app's balance to a user. On success, returns completed *aiosend.types.Transfer*. This method must first be enabled in the security settings of your app.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_c932ff301f

Parameters

- **user_id** (int) – User ID in Telegram. User must have previously used @CryptoBot (@CryptoTestnetBot for testnet).
- **asset** (*Asset*) – Cryptocurrency alphabetic code. Supported assets: "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet).
- **amount** (float) – Amount of the transfer in float. The minimum and maximum amount limits for each of the supported assets roughly correspond to 1-25000 USD. Use *aiosend.CryptoPay.get_exchange_rates()* to convert amounts. For example: *125.50*
- **spend_id** (str | None) – Random UTF-8 string unique per transfer for idempotent requests. The same *spend_id* can be accepted only once from your app. Up to 64 symbols.
- **comment** (str | None) – *Optional*. Comment for the transfer. Users will see this comment in the notification about the transfer. Up to 1024 symbols.

- **disable_send_notification** (bool | None) – *Optional*. Pass true to not send to the user the notification about the transfer. Defaults to `False`.

Return type
Transfer

`CryptoPay.get_invoices(asset=None, fiat=None, invoice_ids=None, status=None, offset=None, count=None)`

getInvoices method.

Use this method to get invoices created by your app. On success, returns array of `aiosend.types.Invoice`.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_e4c2ccb208

Parameters

- **asset** (*Asset* | None) – *Optional*. Cryptocurrency alphabetic code. Supported assets: "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet). Defaults to all currencies.
- **fiat** (*Fiat* | None) – *Optional*. Fiat currency code. Supported fiat currencies: "USD", "EUR", "RUB", "BYN", "UAH", "GBP", "CNY", "KZT", "UZS", "GEL", "TRY", "AMD", "THB", "INR", "BRL", "IDR", "AZN", "AED", "PLN" and "ILS". Defaults to all currencies.
- **invoice_ids** (list[int] | None) – *Optional*. List of invoice IDs separated by comma.
- **status** (Optional[Literal[ACTIVE, PAID]]) – *Optional*. Status of invoices to be returned. Available statuses: "active" and "paid". Defaults to all statuses.
- **offset** (int | None) – *Optional*. Offset needed to return a specific subset of invoices. Defaults to 0.
- **count** (int | None) – *Optional*. Number of invoices to be returned. Values between 1-1000 are accepted. Defaults to 100.

Return type
list[*Invoice*]

`CryptoPay.get_checks(asset=None, check_ids=None, status=None, offset=None, count=None)`

getChecks method.

Use this method to get checks created by your app. On success, returns array of `aiosend.types.Check`.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_d23dda1828

Parameters

- **asset** (*Asset* | None) – *Optional*. Cryptocurrency alphabetic code. Supported assets: "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet). Defaults to all currencies.
- **check_ids** (list[int] | None) – *Optional*. List of check IDs separated by comma.
- **status** (*CheckStatus* | None) – *Optional*. Status of check to be returned. Available statuses: "active" and "activated". Defaults to all statuses.
- **offset** (int | None) – *Optional*. Offset needed to return a specific subset of check. Defaults to 0.
- **count** (int | None) – *Optional*. Number of checks to be returned. Values between 1-1000 are accepted. Defaults to 100.

Return type
list[*Check*]

`CryptoPay.get_transfers(asset=None, transfer_ids=None, spend_id=None, offset=None, count=None)`

getTransfers method.

Use this method to get transfers created by your app. On success, returns array of `aiosend.types.Transfer`.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_a9c71eaebd

Parameters

- **asset** (*Asset* | None) – *Optional*. Cryptocurrency alphabetic code. Supported assets: "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet). Defaults to all currencies.
- **transfer_ids** (list[int] | None) – *Optional*. List of transfer IDs separated by comma.
- **spend_id** (str | None) – *Optional*. Unique UTF-8 transfer string.
- **offset** (int | None) – *Optional*. Offset needed to return a specific subset of transfers. Defaults to 0.
- **count** (int | None) – *Optional*. Number of transfers to be returned. Values between 1-1000 are accepted. Defaults to 100.

Return type

list[*Transfer*]

`CryptoPay.get_balance()`

getBalance method.

Use this method to get balances of your app. Requires no parameters. Returns array of `aiosend.types.Balance`.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_86005049de

Return type

list[*Balance*]

`CryptoPay.get_exchange_rates()`

getExchangeRates method.

Use this method to get exchange rates of supported currencies. Requires no parameters. Returns array of `aiosend.types.ExchangeRate`.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_bc0e2dee1c

Return type

list[*ExchangeRate*]

`CryptoPay.get_currencies()`

getCurrencies method.

Use this method to get a list of supported currencies. Requires no parameters. Returns a list of fiat and cryptocurrency alphabetic codes.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_f5cd57dd47

Return type

list[*Currency*]

`CryptoPay.get_stats(start_at=None, end_at=None)`

getStats method.

Use this method to get app statistics. On success, returns `aiosend.types.AppStats`.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_1590f8f4ed

Parameters

- **start_at** (datetime | None) – *Optional*. Date from which start calculating statistics. Default is current date minus 24 hours.
- **end_at** (datetime | None) – *Optional*. The date on which to finish calculating statistics. Default is current date.

Return type

AppStats

3.2.2 Types

Crypto Pay API types implementation.

class aiosend.types.**App**(***data*)

App object.

This object represents the *Crypto Pay* app.

app_id: int

Crypto Pay app id.

name: str

Crypto Pay app name.

payment_processing_bot_username: str

Telegram username of the payment processing bot.

class aiosend.types.**AppStats**(***data*)

AppStats object.

Source: http://help.crypt.bot/crypto-pay-api#wnPA

conversion: float

Conversion of all created invoices.

created_invoice_count: int

Total created invoice count.

end_at: datetime

The date on which the statistics calculation was ended in ISO 8601 format.

paid_invoice_count: int

Total paid invoice count.

start_at: datetime

The date on which the statistics calculation was started in ISO 8601 format.

unique_users_count: int

The unique number of users who have paid the invoice.

volume: float

Total volume of paid invoices in USD.

class aiosend.types.**Balance**(***data*)

Balance object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_0593a935bb

available: float

Total available amount in float.

currency_code: *Asset* | str

Cryptocurrency alphabetic code. Currently, can be "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet).

onhold: float

Unavailable amount currently is on hold in float.

class aiosend.types.Check(data)**

Check object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_2adb5a31b0

activated_at: datetime | None

Date the check was activated in ISO 8601 format.

amount: float

Amount of the check in float.

asset: *Asset* | str

Cryptocurrency alphabetic code. Currently, can be "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet).

bot_check_url: str

URL should be provided to the user to activate the check.

check_id: int

Unique ID for this check.

created_at: datetime

Date the check was created in ISO 8601 format.

hash: str

Hash of the check.

status: *CheckStatus* | str

Status of the check, can be "active" or "activated".

class aiosend.types.CryptoPayObject(data)**

Base object class for types.

class aiosend.types.Currency(data)**

Currency object.

This object represents an *Crypto Pay* currency.

code: *Fiat* | *Asset* | str

Currency code.

decimals: int

Currency decimals.

is_blockchain: bool

True, if the currency is blockchain currency.

is_fiat: bool

True, if the currency is fiat.

is_stablecoin: bool

True, if the currency is stablecoin.

name: str

Currency name.

url: str | None

Currency url.

class aiosend.types.**Error**(***data*)

API error model.

code: int

name: str

class aiosend.types.**ExchangeRate**(***data*)

ExchangeRate object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_532277de9c

is_crypto: bool

True, if the source is the cryptocurrency.

is_fiat: bool

True, if the source is the fiat currency.

is_valid: bool

True, if the received rate is up-to-date.

rate: float

The current rate of the source asset valued in the target currency.

source: *Asset* | *Fiat* | str

Currency alphabetic code. Currently, can be “USDT”, “TON”, “BTC”, “ETH”, “LTC”, “BNB”, “TRX” and “USDC”.

target: *Fiat* | str

Fiat currency code. Currently, can be “USD”, “EUR”, “RUB”, “BYN”, “UAH”, “GBP”, “CNY”, “KZT”, “UZS”, “GEL”, “TRY”, “AMD”, “THB”, “INR”, “BRL”, “IDR”, “AZN”, “AED”, “PLN” and “ILS”.

class aiosend.types.**Invoice**(***data*)

Invoice object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_bea6645543

accepted_assets: list[*Asset* | str] | None

Optional. List of assets which can be used to pay the invoice. Available only if currency_type is “fiat”. Currently, can be “USDT”, “TON”, “BTC”, “ETH”, “LTC”, “BNB”, “TRX” and “USDC” (and “JET” for testnet).

allow_anonymous: bool

True, if the user can pay the invoice anonymously.

allow_comments: bool

True, if the user can add comment to the payment.

amount: float

Amount of the invoice for which the invoice was created.

asset: *Asset* | str | None

Optional. Cryptocurrency code. Available only if the value of the field currency_type is "crypto". Currently, can be "USDT", "TON", "BTC", "ETH", "LTC", "BNB", "TRX" and "USDC" (and "JET" for testnet).

bot_invoice_url: str

URL should be provided to the user to pay the invoice.

comment: str | None

Optional. Comment to the payment from the user.

created_at: datetime

Date the invoice was created in ISO 8601 format.

currency_type: *CurrencyType* | str

Type of the price, can be "crypto" or "fiat".

description: str | None

Optional. Description for this invoice.

expiration_date: datetime | None

Optional. Date the invoice expires in ISO 8601 format.

fee: str | None

Optional. Amount of charged service fees. Available only in the payload of the webhook update (described here for reference).

fee_amount: float | None

Optional. Amount of service fees charged when the invoice was paid. Available only if status is "paid".

fee_asset: *Asset* | str | None

Optional. Asset of service fees charged when the invoice was paid. Available only if status is "paid".

fee_in_usd: float | None

Optional. Amount in USD of service fees charged when the invoice was paid. Available only if status is "paid".

fiat: *Fiat* | str | None

Optional. Fiat currency code. Available only if the value of the field currency_type is "fiat". Currently one of "USD", "EUR", "RUB", "BYN", "UAH", "GBP", "CNY", "KZT", "UZS", "GEL", "TRY", "AMD", "THB", "INR", "BRL", "IDR", "AZN", "AED", "PLN" and "ILS".

hash: str

Hash of the invoice.

hidden_message: str | None

Optional. Text of the hidden message for this invoice.

invoice_id: int

Unique ID for this invoice.

is_swapped: bool | None

Optional. For invoices with the "paid" status, this flag indicates whether the swap was successful (only applicable if *swap_to* is set).

mini_app_invoice_url: str

Use this URL to pay an invoice to the Telegram Mini App version.

paid_amount: float | None

Optional. Amount of the invoice for which the invoice was paid. Available only if currency_type is "fiat" and status is "paid".

paid_anonymously: bool | None

True, if the invoice was paid anonymously.

paid_asset: *Asset* | str | None

Optional. Cryptocurrency alphabetic code for which the invoice was paid. Available only if currency_type is "fiat" and status is "paid".

paid_at: datetime | None

Optional. Date the invoice was paid in ISO 8601 format.

paid_btn_name: *PaidBtnName* | str | None

Optional. Label of the button, can be "viewItem", "openChannel", "openBot" or "callback".

paid_btn_url: str | None

Optional. URL opened using the button.

paid_fiat_rate: float | None

Optional. The rate of the paid_asset valued in the fiat currency. Available only if the value of the field currency_type is "fiat" and the value of the field status is "paid".

paid_usd_rate: float | None

Optional. Price of the asset in USD. Available only if status is "paid".

pay_url: str | None

Deprecated. URL should be provided to the user to pay the invoice (described here for reference).

payload: str | None

Optional. Previously provided data for this invoice.

status: *InvoiceStatus* | str | None

Status of the transfer, can be "active", "paid" or "expired".

swap_to: *Asset* | str | None

"USDT", "TON", "TRX", "ETH", "SOL", "BTC", "LTC".

Type

Optional. The asset that will be attempted to be swapped into after the user makes a payment (the swap is not guaranteed). Supported assets

swapped_output: float | None

Optional. If *is_swapped* is *True*, stores the amount received as a result of the swap (in the *swapped_to* asset).

swapped_rate: float | None

Optional. If *is_swapped* is *True*, stores the exchange rate at which the swap was executed.

swapped_to: *Asset* | None

Optional. If *is_swapped* is *True*, stores the asset into which the swap was made.

swapped_uid: str | None

Optional. If *is_swapped* is *True*, stores the unique identifier of the swap.

swapped_usd_amount: float | None

Optional. If *is_swapped* is *True*, stores the resulting swap amount in USD.

swapped_usd_rate: float | None

Optional. If *is_swapped* is *True*, stores the USD exchange rate of the currency from *swapped_to*.

usd_rate: str | None

Optional. Price of the asset in USD. Available only in the Webhook update payload.

web_app_invoice_url: str

Use this URL to pay an invoice to the Web version of Crypto Bot.

class aiosend.types.**ItemsList**(***data*)

Items list.

This model is used to convert a dictionary with the *items* key to a list.

items: _CryptoPayType

class aiosend.types.**Response**(***data*)

API response model.

error: *Error* | None

ok: bool

result: _CryptoPayType | *ItemsList* | None

class aiosend.types.**Transfer**(***data*)

Transfer object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_771b6213d1

amount: float

Amount of the transfer in float.

asset: *Asset* | str

Cryptocurrency alphabetic code. Currently, can be “USDT”, “TON”, “BTC”, “ETH”, “LTC”, “BNB”, “TRX” and “USDC” (and “JET” for testnet).

comment: str | None

Optional. Comment for this transfer.

completed_at: datetime

Date the transfer was completed in ISO 8601 format.

spend_id: str | None

Unique UTF-8 string.

status: Literal['completed']

Status of the transfer, can only be “completed”.

transfer_id: int

Unique ID for this transfer.

user_id: int

Telegram user ID the transfer was sent to.

class aiosend.types.**Update**(***data*)

Update object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_20aded801d

payload: *Invoice*

Payload contains the class Invoice.

request_date: datetime

Date the request was sent in ISO 8601 format.

update_id: int

Non-unique update ID.

update_type: *UpdateType*

invoice_paid - the update sent when the invoice is paid.

Type

Webhook update type. Supported update types

3.2.3 Enums

class aiosend.enums.**Asset**(**values*)

Bases: `str`, `Enum`

Cryptocurrency alphabetic code.

BNB = 'BNB'

BONK = 'BONK'

BTC = 'BTC'

CATI = 'CATI'

DOGS = 'DOGS'

ETH = 'ETH'

GRAM = 'GRAM'

HMSTR = 'HMSTR'

JET = 'JET'

LTC = 'LTC'

MAJOR = 'MAJOR'

MELANIA = 'MELANIA'

MEMHASH = 'MEMHASH'

MY = 'MY'

NOT = 'NOT'

PEPE = 'PEPE'

SEND = 'SEND'

SOL = 'SOL'

TON = 'TON'

TRUMP = 'TRUMP'

TRX = 'TRX'

USDC = 'USDC'

USDT = 'USDT'

WIF = 'WIF'

XAUT = 'XAUT'

class aiosend.enums.CheckStatus(*values)

Bases: str, Enum

Status of check.

ACTIVATED = 'activated'

ACTIVE = 'active'

class aiosend.enums.CurrencyType(*values)

Bases: str, Enum

Type of currency.

CRYPTO = 'crypto'

FIAT = 'fiat'

class aiosend.enums.Fiat(*values)

Bases: str, Enum

Fiat currency code.

AED = 'AED'

AMD = 'AMD'

AZN = 'AZN'

BRL = 'BRL'

BYN = 'BYN'

CNY = 'CNY'

EUR = 'EUR'

GBP = 'GBP'

GEL = 'GEL'

IDR = 'IDR'

ILS = 'ILS'

INR = 'INR'

KGS = 'KGS'

KZT = 'KZT'

PLN = 'PLN'

RUB = 'RUB'

THB = 'THB'

TJS = 'TJS'

TRY = 'TRY'

UAH = 'UAH'

USD = 'USD'

UZS = 'UZS'

class aiosend.enums.**InvoiceStatus**(**values*)

Bases: str, Enum

Status of invoice.

ACTIVE = 'active'

EXPIRED = 'expired'

PAID = 'paid'

class aiosend.enums.**PaidBtnName**(**values*)

Bases: str, Enum

Paid button name.

Label of the button which will be presented to a user after the invoice is paid.

CALLBACK = 'callback'

OPENBOT = 'openBot'

OPENCHANNEL = 'openChannel'

VIEWITEM = 'viewItem'

class aiosend.enums.**UpdateType**(**values*)

Bases: str, Enum

Webhook update type.

INVOICE_PAID = 'invoice_paid'

3.2.4 Network

Network is a class allows you to build a URL to the API endpoints for specific network.

Crypto Pay API has two networks:

- aiosend.**MAINNET**

  alias of Network(name='MAINNET', base='https://pay.crypt.bot/api/{method}')

The main network for transactions with real cryptocurrency.

- `aiosend.TESTNET`

  alias of Network(name='TESTNET', base='https://testnet-pay.crypt.bot/api/{method}')

The network for testing purposes in which currency has no real value.

Usage example

You can create an instance of `CryptoPay` client for both nets.

```
from aiosend import MAINNET, TESTNET, CryptoPay

main_client = CryptoPay(
    "TOKEN",
    MAINNET,  # MAINNET is default
)

test_client = CryptoPay(
    "TOKEN",
    TESTNET,
)
```

class `aiosend.client.network.Network`(*name*, *base*)

Configuration for endpoints.

base: str

Base URL

get_check_image(*asset*, *asset_amount*, *fiat*, *fiat_amount*, *main*)

Return check image url.

Return type

`str`

get_qr(*link*)

Return qr code link.

Return type

`str`

get_rates_image(*base*, *quote*, *rate*, *percent*)

Return rates image url.

Return type

`str`

name: str

Net name

url(*method*)

Return URL for method.

Return type

`str`

3.2.5 Errors

exception aiosend.exceptions.**APIError**(*method*, *error*)

Bases: *CryptoPayError*

Exception for API errors.

exception aiosend.exceptions.**APITimeoutError**(*method*, *timeout*)

Bases: *CryptoPayError*

Exception raised when the API request times out.

exception aiosend.exceptions.**CryptoPayError**

Bases: Exception

Base class for exceptions in this module.

exception aiosend.exceptions.**DeserializationError**(*method*, *message*)

Bases: *CryptoPayError*

Exception for deserialization errors.

exception aiosend.exceptions.**MethodValuesError**(*message*)

Bases: *CryptoPayError*

Exception raised when method values are invalid.

exception aiosend.exceptions.**WrongNetworkError**(*message*)

Bases: *CryptoPayError*

Exception raised when the token is served by different network.

3.3 Client

aiosend is python implementation for Crypto Pay API.

- **Tools** - additional functionality.
- **Shortcut methods** - shortcut type methods.
- **Session** - http session implementation.

3.3.1 Tools

Additional functionality.

CryptoPay.**exchange**(*amount*, *source*, *target*)

Exchange currency.

Wrapper for *aiosend.CryptoPay.get_exchange_rates()*.

Use this method to get the equivalent amount in the target currency for the source currency.

Return type
float

Raise
aiosend.exceptions.CryptoPayError if there is no such exchange rate.

Usage example

```python
import asyncio
from aiosend import CryptoPay

async def main() -> None:
    cp = CryptoPay(token="TOKEN")
    print(await cp.exchange(10, "USDT", "USD"))  # 9.998635

if __name__ == "__main__":
    asyncio.run(main())
```

CryptoPay.**get_balance_by_asset**(*asset*)

Get the balance of a specific asset.

Wrapper for *aiosend.CryptoPay.get_balance()*.

Use this method to get the balance of a specific asset.

Return type
Balance

Raise
aiosend.exceptions.CryptoPayError if there is no such asset.

Usage example

```python
import asyncio
from aiosend import CryptoPay

async def main() -> None:
    cp = CryptoPay(token="TOKEN")
    print(await cp.get_balance_by_asset("USDT"))  # 1.2345

if __name__ == "__main__":
    asyncio.run(main())
```

CryptoPay.**get_invoice**(*invoice*)

Get exactly one invoice or none.

Wrapper for *aiosend.CryptoPay.get_invoices()*

Use this method to update status of an existing invoice object or to get this object by passing the invoice id.

Return type
Invoice | None

Usage example

```python
import asyncio
from aiosend import CryptoPay

async def main() -> None:
    cp = CryptoPay(token="TOKEN")

    invoice = await cp.create_invoice(1, "TON")
    print(invoice.status)  # active
    await asyncio.sleep(10)  # payment
    new_invoice = await cp.get_invoice(invoice)
```

(continues on next page)

(continued from previous page)

```
    print(new_invoice.status)  # paid

if __name__ == "__main__":
    asyncio.run(main())
```

`CryptoPay.get_check(check)`

Get exactly one check or none.

Wrapper for *aiosend.CryptoPay.get_checks()*

Use this method to update status of an existing check object or to get this object by passing the check id.

Return type
Check | None

`CryptoPay.delete_all_checks()`

Delete all checks.

Wrapper for *aiosend.CryptoPay.get_checks()* and *aiosend.CryptoPay.delete_check()*

Use this method to delete all existing checks created by your app.

Return type
None

`CryptoPay.delete_all_invoices()`

Delete all invoices.

Wrapper for *aiosend.CryptoPay.get_invoices()* and *aiosend.CryptoPay.delete_invoice()*

Use this method to delete all existing invoices created by your app.

Return type
None

`CryptoPay.get_rates_image(base, quote, rate, percent)`

Get rates image.

Use this method to get exchange rates image.

Return type
str

3.3.2 Shortcut methods

These methods simplify the use of standard *Crypto Pay API methods* and also provide additional functionality to the objects.

Updating objects

`Balance.update()`

Shortcut for method *aiosend.CryptoPay.get_balance()*.

Use this method to update balance object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_86005049de

Return type
None

`Check.update()`

Shortcut for method *aiosend.CryptoPay.get_checks()*.

Use this method to update check object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_d23dda1828

Return type
None

`ExchangeRate.update()`

Shortcut for method *aiosend.CryptoPay.get_exchange_rates()*.

Use this method to update ExchangeRate object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_bc0e2dee1c

Return type
None

`Invoice.update()`

Shortcut for method *aiosend.CryptoPay.get_invoices()*.

Use this method to update invoice object.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_e4c2ccb208

Return type
None

Deleting objects

`Check.delete()`

Shortcut for method *aiosend.CryptoPay.delete_check()*.

Use this method to delete check created by your app. Returns `True` on success.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_b27428d56a

Return type
bool

`Invoice.delete()`

Shortcut for method *aiosend.CryptoPay.delete_invoice()*.

Use this method to delete invoice created by your app. Returns `True` on success.

Source: https://help.send.tg/en/articles/10279948-crypto-pay-api#h_da2ea9f39c

Return type
bool

QR code of object

`property Check.qr: str`

Get check qr code.

`property Invoice.qr: str`

Get invoice qr code.

Check image

`Check.get_image(fiat)`

Get check preview image.

Return type
`str`

Poll objects

`Check.poll(**kwargs)`

Send the check to the polling manager.

Use this method to check the status of the check until the timeout expires.

Parameters
kwargs (`object`) – additional payload for the handler.

Return type
`None`

`Invoice.poll(**kwargs)`

Send the invoice to the polling manager.

Use this method to check the status of the invoice until the timeout expires.

Parameters
kwargs (`object`) – additional payload for the handler.

Return type
`None`

Usage examples

```
import asyncio
from aiosend import CryptoPay

async def main() -> None:
    cp = CryptoPay("TOKEN")

    invoice = await cp.create_invoice(1, "USDT")

    print(invoice.status)  # active
    await asyncio.sleep(10)  # payment
    await invoice.update()
    print(invoice.status)  # paid

    print(invoice.qr)  # qr code link

    await invoice.delete()  # delete the invoice

if __name__ == "__main__":
    asyncio.run(main())
```

```
import asyncio
from aiosend import CryptoPay

```

(continues on next page)

(continued from previous page)

```
async def main() -> None:
    cp = CryptoPay("TOKEN")

    check = await cp.create_check(1, "USDT")

    # invoice preview image with fiat conversion
    print(await check.get_image("USD"))

    print(check.qr)  # check qr code link

    await check.delete()  # delete check

if __name__ == "__main__":
    asyncio.run(main())
```

3.3.3 Dependency Injection

Dependency injection is a mechanism that allows you to make the program more flexible, adaptable, and extensible by decoupling the usage of an object from its creation.

Usage example

```
import asyncio
from aiosend import CryptoPay
from aiosend.types import Invoice

cp = CryptoPay("TOKEN", var1="value1")  # injection
cp["var2"] = "value2"  # injection

async def main() -> None:
    invoice = await cp.create_invoice(1, "USDT")
    print("invoice link:", invoice.bot_invoice_url)
    invoice.poll(var3="value3")  # injection
    await cp.start_polling()

@cp.invoice_paid()
async def payment_handler(
    invoice: Invoice,
    var1: str,  # this one is from constructor
    var2: str,  # this one is from key assignment
    var3: str,  # this one is from poll method kwargs
) -> None:
    print(f"Invoice #{invoice.invoice_id} {var1=} {var2=} {var3=}")

if __name__ == "__main__":
    asyncio.run(main())
```

3.3.4 Type hints and warnings

Literal type hints

aiosend provides rich typing that allows you not to remember the values of the enumerated parameters.

```
cp.create_check(1, "")
"BNB"
"BONK"
"BTC"
"CATI"
"DOGS"
"ETH"
"GRAM"
"HMSTR"
"JET"
"LTC"
"MAJOR"
"MELANIA"
```

```
cp.create_invoice(1, "USDT", paid_btn_name="")
"callback"
"openBot"
"openChannel"
"viewItem"
```

Wrong net warning

aiosend will warn you if you are using the wrong net.

```
from aiosend import CryptoPay, TESTNET
cp = CryptoPay("MAINNET token", TESTNET)
```

```
aiosend.exceptions.WrongNetworkError: Authorization failed.
Token is served by the MAINNET, you are using TESTNET
```

3.3.5 Session

aiosend uses aiohttp session by default. You can implement your own session by inheriting `BaseSession` and overriding `request` method.

`class aiosend.client.session.AiohttpSession`(*network*, *timeout=300*)

Bases: `BaseSession`

Http session based on aiohttp.

This class is a wrapper of *aiohttp.ClientSession*.

`async request`(*token*, *client*, *method*)

Make http request.

Return type

`TypeVar`(`_CryptoPayType`, bound= CryptoPayObject | list | bool)

`class aiosend.client.session.BaseSession`(*network*, *timeout=300*)

Bases: `ABC`

Abstract session class.

If you want to implement your own session class, you should inherit this class.

`abstractmethod async request`(*token*, *client*, *method*)

Make http request.

Return type

`TypeVar`(`_CryptoPayType`, bound= CryptoPayObject | list | bool)

3.4 Events

- **Webhook** - updates handling via webhook.
- **Invoice polling** - invoice updates handling via polling.
- **Check polling** - check updates handling via polling.
- **Polling config** - configuration for update requests

3.4.1 Routers

Router is an object for routing updates. There are two types of routers: `PollingRouter` and `WebhookRouter`. `CryptoPay` is the root router for both of them. One router can include another router, but in the end, the parent router must be included in the root router via `include_router` or `include_routers` methods.

Usage example

```python
from aiosend import PollingRouter
from aiosend.types import Invoice

router = PollingRouter(name=__name__)

@router.invoice_paid()
def invoice_paid(invoice: Invoice) -> None:
    print(f"invoice_paid: {invoice.invoice_id}")
```

class aiosend._events.**BaseRouter**(*, *name=None*)

Base router for handling and propagate events.

include_router(*router*)

Include another router to this one.

Return type
None

include_routers(**routers*)

Include multiple routers to this one.

Return type
None

class aiosend.polling.**PollingRouter**(*, *name=None*)

Bases: `BaseRouter`

Router for polling events.

class aiosend.webhook.**WebhookRouter**(*, *name=None*)

Bases: `BaseRouter`

Router for webhook updates.

3.4.2 Event observers

Event observer is an object that stores event handlers. You can attach a new event handler to event observer using a `@router.<event type>(...)` decorator or a `router.<event type>.register(...)` method.

class aiosend._events.**EventObserver**

Event observer for storing handlers and propagating events.

register(*handler*, **filters*)

Register event handler.

Return type
None

async **trigger**(*event*, ***kwargs*)

Trigger event observer.

Return type
bool

Polling Router

Available observers for `PollingRouter` are listed here.

Paid invoice

```
@router.invoice_paid()
def invoice_paid(invoice: Invoice) -> None: ...
```

Expired invoice

```
@router.invoice_expired()
def invoice_expired(invoice: Invoice) -> None: ...
```

Activated check

```
@router.check_activated()
def check_activated(check: Check) -> None: ...
```

Expired check

```
@router.check_expired()
def check_expired(check: Check) -> None: ...
```

Webhook Router

Available observers for *WebhookRouter* are listed here.

Paid invoice

```
@router.invoice_paid()
def invoice_paid(invoice: Invoice) -> None: ...
```

3.4.3 Webhook

Tip

To use webhooks you need to enable them in the @CryptoBot settings as follows:

Crypto Pay -> My Apps -> YOUR APP -> Webhooks -> Enable webhooks.

Usage example with aiohttp web server

```
import asyncio
from aiohttp.web import Application, _run_app
from aiosend import CryptoPay
from aiosend.types import Invoice
from aiosend.webhook import AiohttpManager

app = Application()
cp = CryptoPay(
    "TOKEN",
    webhook_manager=AiohttpManager(app, "/handler"),
)

@cp.invoice_paid()
async def handler(invoice: Invoice) -> None:
    print(f"Received {invoice.amount} {invoice.asset}")
```

(continues on next page)

(continued from previous page)

```
async def main() -> None:
    invoice = await cp.create_invoice(1, "USDT")
    print("invoice link:", invoice.bot_invoice_url)
    await _run_app(app)

if __name__ == "__main__":
    asyncio.run(main())
```

Usage example with fastapi web server

Tip

In order to use aiosend with fastapi you need to install extra package

```
pip install aiosend[fastapi]
```

```
import asyncio
import uvicorn
from fastapi import FastAPI
from aiosend import CryptoPay
from aiosend.types import Invoice
from aiosend.webhook import FastAPIManager

app = FastAPI()
cp = CryptoPay(
    "TOKEN",
    webhook_manager=FastAPIManager(app, "/handler"),
)

@cp.invoice_paid()
async def handler(invoice: Invoice) -> None:
    print(f"Received {invoice.amount} {invoice.asset}")

async def main() -> None:
    invoice = await cp.create_invoice(1, "USDT")
    print("invoice link:", invoice.bot_invoice_url)

if __name__ == "__main__":
    asyncio.run(main())
    uvicorn.run(app)
```

Usage example with flask web server

Tip

In order to use aiosend with flask you need to install extra package

```
pip install aiosend[flask]
```

```python
import asyncio
from flask import Flask
from aiosend import CryptoPay
from aiosend.types import Invoice
from aiosend.webhook import FlaskManager

app = Flask(__name__)
cp = CryptoPay(
    "TOKEN",
    webhook_manager=FlaskManager(app, "/handler"),
)

@cp.invoice_paid()
async def handler(invoice: Invoice) -> None:
    print(f"Received {invoice.amount} {invoice.asset}")

async def main() -> None:
    invoice = await cp.create_invoice(1, "USDT")
    print("invoice link:", invoice.bot_invoice_url)

if __name__ == "__main__":
    asyncio.run(main())
    app.run()
```

aiosend uses aiohttp as web server by default. You can implement your own webhook manager by inheriting *aiosend.webhook.WebhookManager* and overriding *aiosend.webhook.WebhookManager.register_handler*.

`class aiosend.webhook.`**`WebhookManager`**(*app*, *path*)

Webhook manager.

If you want to implement your own webhook manager, you must inherit from this class.

`abstractmethod` **`register_handler`**(*feed_update*)

Register webhook handler.

Override this method in your own webhook manager class. This method is used for registering webhook handler in your app.

Parameters

`handler` – Web server handler object.

Return type

`None`

`class aiosend.webhook.`**`AiohttpManager`**(*app*, *path*)

Bases: *WebhookManager*[`Application`]

aiohttp webhook manager.

Webhook manager based on `aiohttp.web.Application`.

`register_handler`(*feed_update*)

Register webhook handler.

Return type

`None`

class aiosend.webhook.**FastAPIManager**(*app*, *path*)

Bases: *WebhookManager*[FastAPI | APIRouter]

FastAPI webhook manager.

Webhook manager based on `fastapi.FastAPI`.

register_handler(*feed_update*)

Register webhook handler.

Return type
None

class aiosend.webhook.**FlaskManager**(*app*, *path*)

Bases: *WebhookManager*[Flask]

Flask webhook manager.

Webhook manager based on `flask.Flask`.

register_handler(*feed_update*)

Register webhook handler.

Return type
None

3.4.4 Invoice polling

Polling is a method of receiving updates by periodically sending requests. Once *invoice status* is changed to *PAID*, *polling manager* will call the `invoice_paid` handler. Invoice polling uses the */getInvoices* method.

> **Attention**
>
> *Polling manager* has *configuration* that defines the *delay* (between requests) and *timeout* for each invoice in the awaiting queue. After the timeout polling manager will stop polling that invoice and call the `invoice_expired` handler if it is declared.
>
> **Default is 2 seconds delay and 300 seconds (5 min) timeout**.
>
> *You can change the polling configuration.*

Usage example

```
import asyncio
from aiosend import CryptoPay
from aiosend.types import Invoice

cp = CryptoPay("TOKEN")

@cp.invoice_paid()
async def payment_handler(invoice: Invoice, payload: str) -> None:
    print("Received", invoice.amount, invoice.asset, payload)

@cp.invoice_expired()
async def expired_invoice_handler(invoice: Invoice, payload: str) -> None:
    print("Expired invoice", invoice.invoice_id, payload)
```

(continues on next page)

(continued from previous page)

```
async def main() -> None:
    invoice = await cp.create_invoice(1, "USDT")
    print("invoice link:", invoice.bot_invoice_url)
    invoice.poll(payload="payload")
    await cp.start_polling()

if __name__ == "__main__":
    asyncio.run(main())
```

3.4.5 Check polling

Polling is a method of receiving updates by periodically sending requests. Once *check status* is changed to *ACTIVATED*, *polling manager* will call the `check_activated` handler. Check polling uses the */getChecks* method.

Attention

Polling manager has *configuration* that defines the *delay* (between requests) and *timeout* for each check in the awaiting queue. After the timeout polling manager will stop polling that check and call the `check_expired` handler if it is declared.

Default is 2 seconds delay and 300 seconds (5 min) timeout.

You can change the polling configuration.

Usage example

```
import asyncio
from aiosend import CryptoPay
from aiosend.types import Check

cp = CryptoPay("TOKEN")

@cp.check_activated()
async def check_handler(check: Check, payload: str) -> None:
    print("Received", check.amount, check.asset, payload)

@cp.check_expired()
async def expired_check_handler(check: Check, payload: str) -> None:
    print("Expired check", check.check_id, payload)

async def main() -> None:
    check = await cp.create_check(1, "USDT")
    print("check link:", check.check_id)
    check.poll(payload="payload")
    await cp.start_polling()

if __name__ == "__main__":
    asyncio.run(main())
```

3.4.6 Filters

Filters are needed for routing updates to the specific handler. *WebhookManager* and *PollingManager* will check if the handler is suitable for the update. Once a handler with a suitable set of filters is found, searching of handler will be stopped.

Filters can be:

- asynchronous function
- synchronous function
- lambda function
- class with a synchronous *__call__* method
- class with an asynchronous *__call__* method
- instance of MagicFilter

Usage example

For example, you can create a couple of invoices and poll them like that

```
import asyncio
from aiosend import CryptoPay
from aiosend.types import Invoice

cp = CryptoPay("TOKEN")

async def main() -> None:
    invoice1 = await cp.create_invoice(
        1, "USDT", payload="product1",
    )
    invoice2 = await cp.create_invoice(
        1, "USDT", payload="product2",
    )
    print(invoice1.bot_invoice_url)
    print(invoice2.bot_invoice_url)
    invoice1.poll()
    invoice2.poll()
    await cp.start_polling()

if __name__ == "__main__":
    asyncio.run(main())
```

Magic filter

```
from magic_filter import F # or from aiogram import F

@cp.invoice_paid(F.payload == "product1")
async def handler1(invoice: Invoice) -> None:
    print(f"paid {invoice.amount} {invoice.asset} for product1")


@cp.invoice_paid(F.payload == "product2")
```

(continues on next page)

(continued from previous page)

```
async def handler2(invoice: Invoice) -> None:
    print(f"paid {invoice.amount} {invoice.asset} for product2")
```

Function filter

You can use either `def`, `async def` or `lambda`.

```
def filter1(invoice: Invoice) -> bool:
    return invoice.payload == "product1"

async def filter2(invoice: Invoice) -> bool:
    return invoice.payload == "product2"

@cp.invoice_paid(filter1)
async def handler1(invoice: Invoice) -> None:
    print(f"paid {invoice.amount} {invoice.asset} for product1")

@cp.invoice_paid(filter2, lambda inv: inv.amount == 1)
async def handler2(invoice: Invoice) -> None:
    print(f"paid {invoice.amount} {invoice.asset} for product2")
```

Class filter

You can use classes that implement either a synchronous (`def`) or an asynchronous (`async def`) *__call__* method.

```
class MySyncFilter:
    def __init__(self, payload: str):
        self.payload = payload

    def __call__(self, invoice: Invoice) -> bool:
        return invoice.payload == self.payload

class MyASyncFilter:
    def __init__(self, payload: str):
        self.payload = payload

    async def __call__(self, invoice: Invoice) -> bool:
        return invoice.payload == self.payload

@cp.invoice_paid(MySyncFilter("product1"))
async def handler1(invoice: Invoice) -> None:
    print(f"paid {invoice.amount} {invoice.asset} for product1")


@cp.invoice_paid(MyASyncFilter("product2"))
async def handler2(invoice: Invoice) -> None:
    print(f"paid {invoice.amount} {invoice.asset} for product2")
```

Get filter result as handler argument

You can use aiogram 3.x magic filter's `as_` method to get filter result as handler argument

```
from magic_filter import F # or from aiogram import F

@cp.invoice_paid(F.payload.as_("payload"))
async def handler1(invoice: Invoice, payload: str) -> None:
    print(f"paid #{invoice.invoice_id} paylaod: {payload}")
```

You can also return context data from any filters like that

```
def myfilter(invoice: Invoice) -> bool | dict[str, object]:
    if invoice.payload is None:
        return False
    return {"payload": invoice.payload}

@cp.invoice_paid(myfilter)
async def handler1(invoice: Invoice, payload: str) -> None:
    print(f"paid #{invoice.invoice_id} paylaod: {payload}")
```

3.4.7 Polling configuration

You can configure your own polling configuration.

class `aiosend.polling.PollingConfig`(*timeout=300*, *delay=2*)

Polling configuration.

delay: int

Time to wait between the requests in seconds.

timeout: int

Timeout in seconds.

```
import asyncio
from aiosend import CryptoPay
from aiosend.polling import PollingConfig
from aiosend.types import Invoice

cp = CryptoPay(
    "TOKEN",
    polling_config=PollingConfig(
        timeout=600,  # 10 minutes
        delay=3,  # request every 3 seconds
    ),
)

@cp.invoice_paid()
async def handler(invoice: Invoice) -> None:
    print("Received", invoice.amount, invoice.asset)

# called after timeout (600s) or when invoice status is "expired"
@cp.invoice_expired()
async def expired_invoice_handler(invoice: Invoice, payload: str) -> None:
    print("Expired invoice", invoice.invoice_id, payload)
```

(continues on next page)

(continued from previous page)

```
async def main() -> None:
    invoice = await cp.create_invoice(1, "USDT")
    print("invoice link:", invoice.bot_invoice_url)
    invoice.poll()
    await cp.start_polling()

if __name__ == "__main__":
    asyncio.run(main())
```

class aiosend.polling.**PollingManager**(*config*)

Polling manager class.

This class is used to handle payments and check activation via polling method.

abstractmethod **get_checks**(*asset=None*, *check_ids=None*, *status=None*, *offset=None*, *count=None*)

getchecks method.

Return type
list[*Check*]

abstractmethod **get_invoices**(*asset=None*, *fiat=None*, *invoice_ids=None*, *status=None*, *offset=None*, *count=None*)

getinvoices method.

Return type
list[*Invoice*]

start_polling(*parallel=None*)

Run polling.

Parameters
parallel (Callable[[], Any] | None) – function to run in background.

Return type
None

3.4.8 Payload Data

Payload Data is used to generate a structured payload using the pydantic model.

class aiosend.**PayloadData**(***data*)

Base payload data class.

classmethod **filter**(*rule=None*)

Generate a filter for payload with rule.

Parameters
rule (MagicFilter | None) – magic rule

Return type
PayloadDataFilter

pack()

Generate payload data string.

Return type
str

classmethod unpack(*value*)

Parse payload data string.

Return type
Self

Usage example

Define a subclass of *PayloadData*. The keyword `prefix` is required to specify the prefix, and the `sep` argument can be provided to define the separator (default is :).

```
class MyData(PayloadData, prefix="md"):
    foo: str
    bar: int
```

Now you can create an instance of this class, pack it into a string, and then pass it in the *Invoice payload*.

```
my_data = MyData(foo="foo", bar=123).pack()
await cp.create_invoice(1, "USDT", payload=my_data)
```

To handle an invoice payment event, you need to declare a handler with a PayloadData filter. You can access an instance of your PayloadData via the handler's named argument `payload_data`.

```
@cp.invoice_paid(MyData.filter())
async def handler(invoice: Invoice, payload_data: MyData) -> None:
    print(invoice.amount, payload_data.foo)
```

Also you can filter events by specific rules using magic filters from aiogram 3.x.

```
from magic_filter import F # or from aiogram import F

@cp.invoice_paid(MyData.filter(F.bar == 123))
async def handler(invoice: Invoice, payload_data: MyData) -> None:
    print(invoice.amount, payload_data.foo)
```

3.5 Integration examples

Examples of usage with libraries are presented here.

3.5.1 aiogram 3.x

Usage example with aiogram 3.x

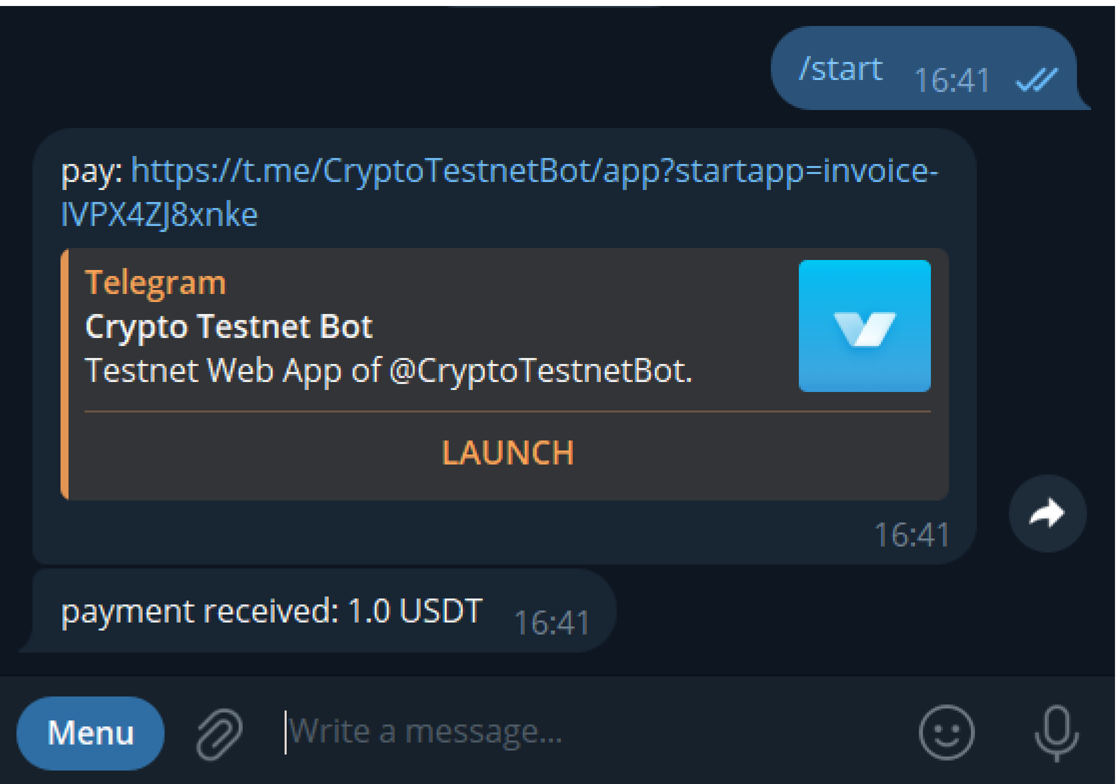
```
import asyncio
from aiogram import Bot, Dispatcher
from aiogram.types import Message
from aiosend import CryptoPay
from aiosend.types import Invoice

cp = CryptoPay("TOKEN")
bot = Bot("TOKEN")
dp = Dispatcher()

@dp.message()
```

(continues on next page)

(continued from previous page)

```
async def get_invoice(message: Message) -> None:
    invoice = await cp.create_invoice(1, "USDT")
    await message.answer(f"pay: {invoice.mini_app_invoice_url}")
    invoice.poll(message=message)

@cp.invoice_paid()
async def handle_payment(
    invoice: Invoice,
    message: Message,
) -> None:
    await message.answer(
        f"payment received: {invoice.amount} {invoice.asset}",
    )

async def main() -> None:
    await asyncio.gather(
        dp.start_polling(bot),
        cp.start_polling(),
    )

if __name__ == "__main__":
    asyncio.run(main())
```

Preview

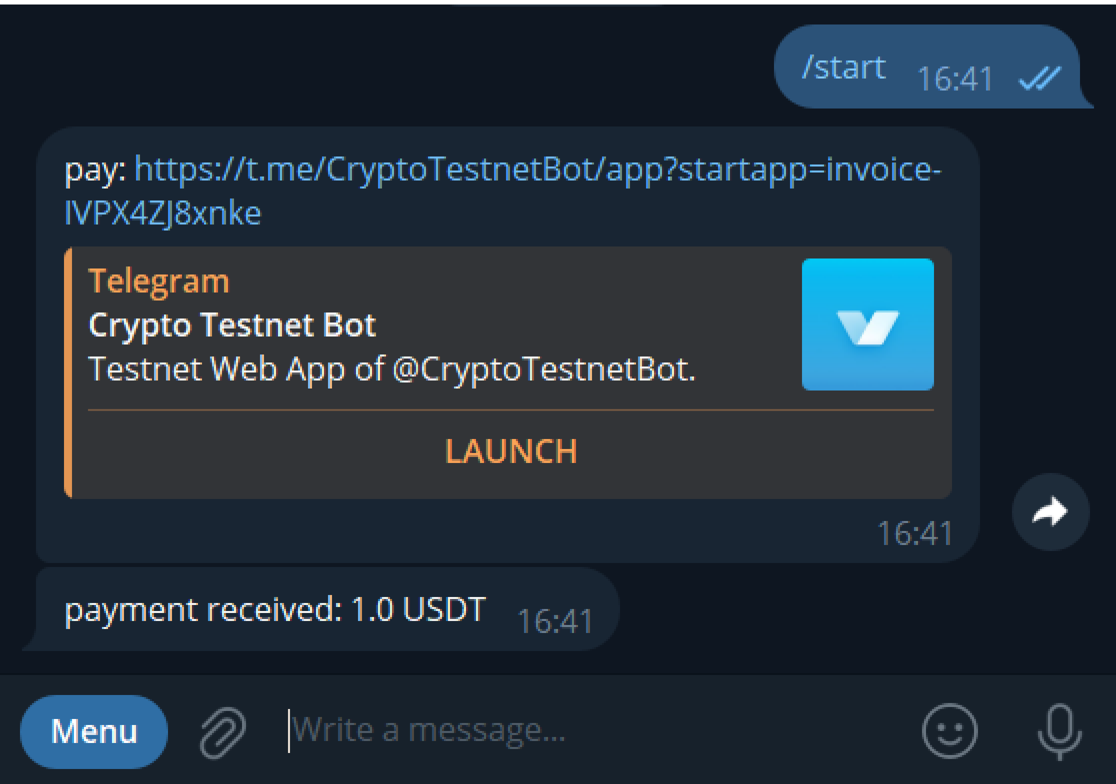


3.5.2 aiogram 2.x

Usage example with aiogram 2.x

```
import asyncio
from aiogram import Bot, Dispatcher, executor
from aiogram.types import Message
from aiosend import CryptoPay
from aiosend.types import Invoice

cp = CryptoPay("TOKEN")
bot = Bot("TOKEN")
dp = Dispatcher(bot)

@dp.message_handler()
async def get_invoice(message: Message) -> None:
    invoice = await cp.create_invoice(1, "USDT")
    await message.answer(f"pay: {invoice.mini_app_invoice_url}")
    invoice.poll(message=message)

@cp.invoice_paid()
async def handle_payment(
    invoice: Invoice,
    message: Message,
) -> None:
    await message.answer(
        f"payment received: {invoice.amount} {invoice.asset}",
    )

async def on_startup(_) -> None:
    asyncio.create_task(cp.start_polling())

if __name__ == "__main__":
    executor.start_polling(dp, on_startup=on_startup)
```

Preview



3.5.3 pyTelegramBotAPI

Usage example with pyTelegramBotAPI

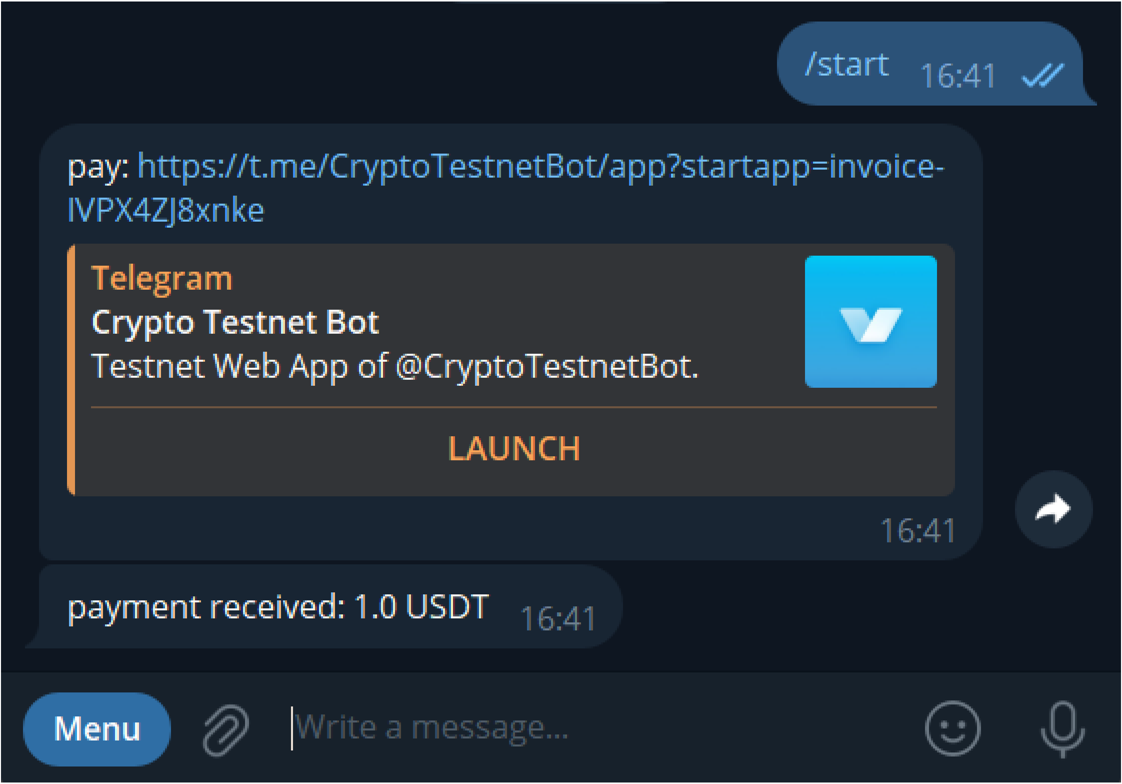
```
from telebot import TeleBot
from telebot.types import Message
from aiosend import CryptoPay
from aiosend.types import Invoice

cp = CryptoPay("TOKEN")
bot = TeleBot("TOKEN")

@bot.message_handler()
def get_invoice(message: Message) -> None:
    invoice = cp.create_invoice(1, "USDT")
    bot.send_message(
        message.from_user.id,
        f"pay: {invoice.mini_app_invoice_url}",
    )
    invoice.poll(user_id=message.from_user.id)

@cp.invoice_paid()
def handle_payment(
    invoice: Invoice,
    user_id: int,
) -> None:
```

(continues on next page)

(continued from previous page)

```
    bot.send_message(
        user_id,
        f"payment received: {invoice.amount} {invoice.asset}",
    )

if __name__ == "__main__":
    cp.start_polling(bot.infinity_polling)
```

Preview



3.5.4 pyTelegramBotAPI (async)

Usage example with async pyTelegramBotAPI

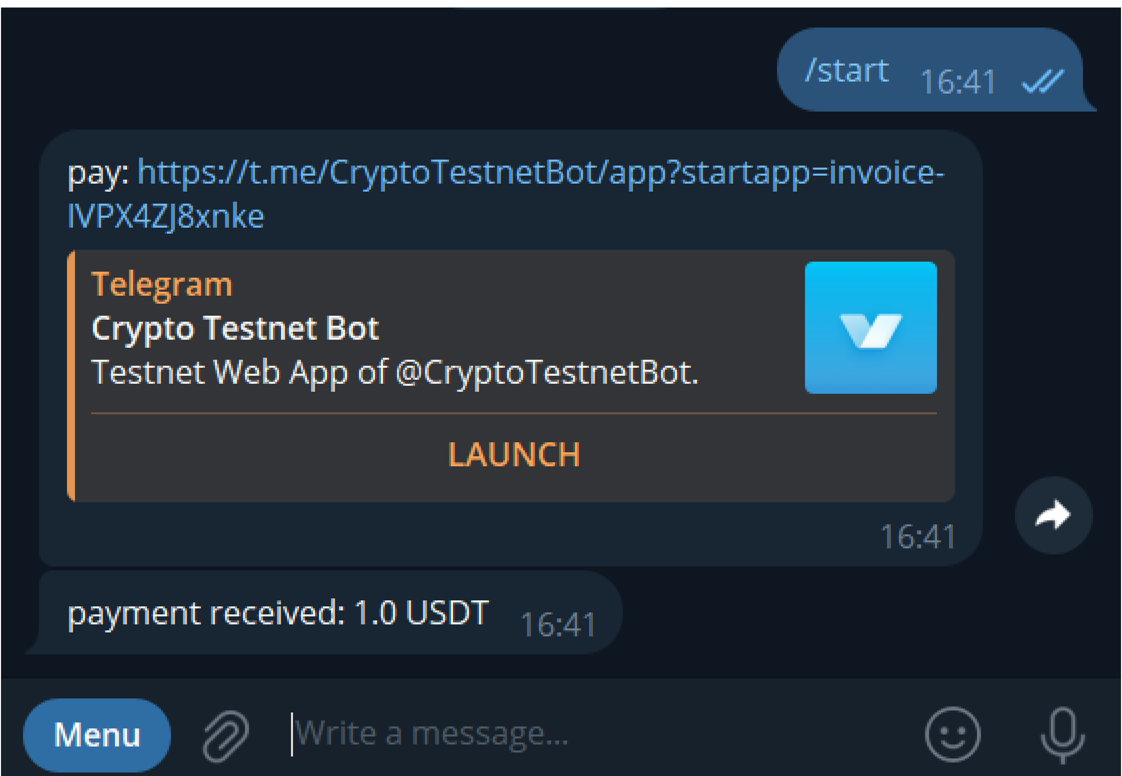
```
import asyncio
from telebot.async_telebot import AsyncTeleBot
from telebot.types import Message
from aiosend import CryptoPay
from aiosend.types import Invoice

cp = CryptoPay("TOKEN")
bot = AsyncTeleBot("TOKEN")

@bot.message_handler()
async def get_invoice(message: Message) -> None:
    invoice = cp.create_invoice(1, "USDT")
```

(continues on next page)

(continued from previous page)

```
    await bot.send_message(
        message.from_user.id,
        f"pay: {invoice.mini_app_invoice_url}",
    )
    invoice.poll(user_id=message.from_user.id)

@cp.invoice_paid()
async def handle_payment(
    invoice: Invoice,
    user_id: int,
) -> None:
    await bot.send_message(
        user_id,
        f"payment received: {invoice.amount} {invoice.asset}",
    )

async def main() -> None:
    await asyncio.gather(
        bot.infinity_polling(),
        cp.start_polling(),
    )

if __name__ == "__main__":
    asyncio.run(main())
```

Preview

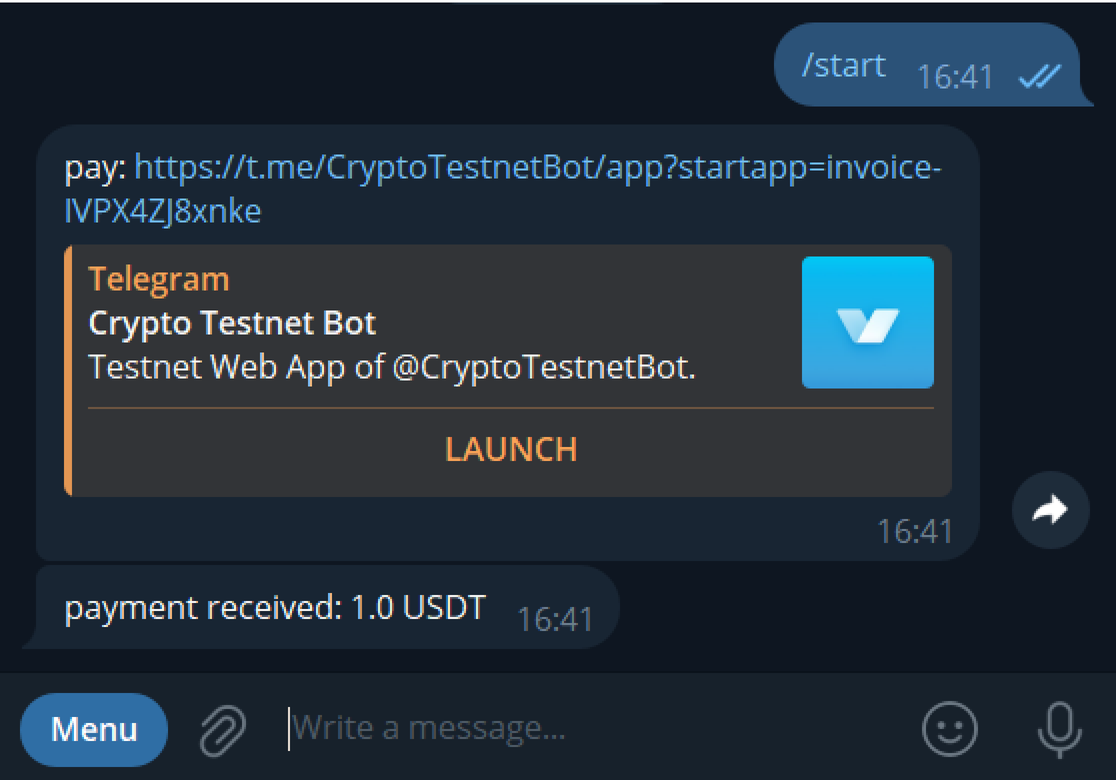


3.5.5 python-telegram-bot

Usage example with python-telegram-bot

```
import asyncio
from telegram import Message, Update
from telegram.ext import (
    Application,
    ContextTypes,
    MessageHandler,
    filters,
)
from aiosend import CryptoPay
from aiosend.types import Invoice

cp = CryptoPay("TOKEN")
app = Application.builder().token("TOKEN").build()

async def get_invoice(
    update: Update,
    context: ContextTypes.DEFAULT_TYPE,
) -> None:
    invoice = await cp.create_invoice(1, "USDT")
    await update.message.reply_text(f"pay: {invoice.mini_app_invoice_url}")
    invoice.poll(message=update.message)

@cp.invoice_paid()
async def handle_payment(
    invoice: Invoice,
    message: Message,
) -> None:
    await message.reply_text(
        f"payment received: {invoice.amount} {invoice.asset}",
    )

async def main() -> None:
    app.add_handler(MessageHandler(filters.TEXT, get_invoice))
    await app.initialize()
    await app.start()
    await asyncio.gather(
        app.updater.start_polling(allowed_updates=Update.ALL_TYPES),
        cp.start_polling(),
    )

if __name__ == "__main__":
    asyncio.run(main())
```

Preview



3.5.6 Pyrogram

Usage example with Pyrogram

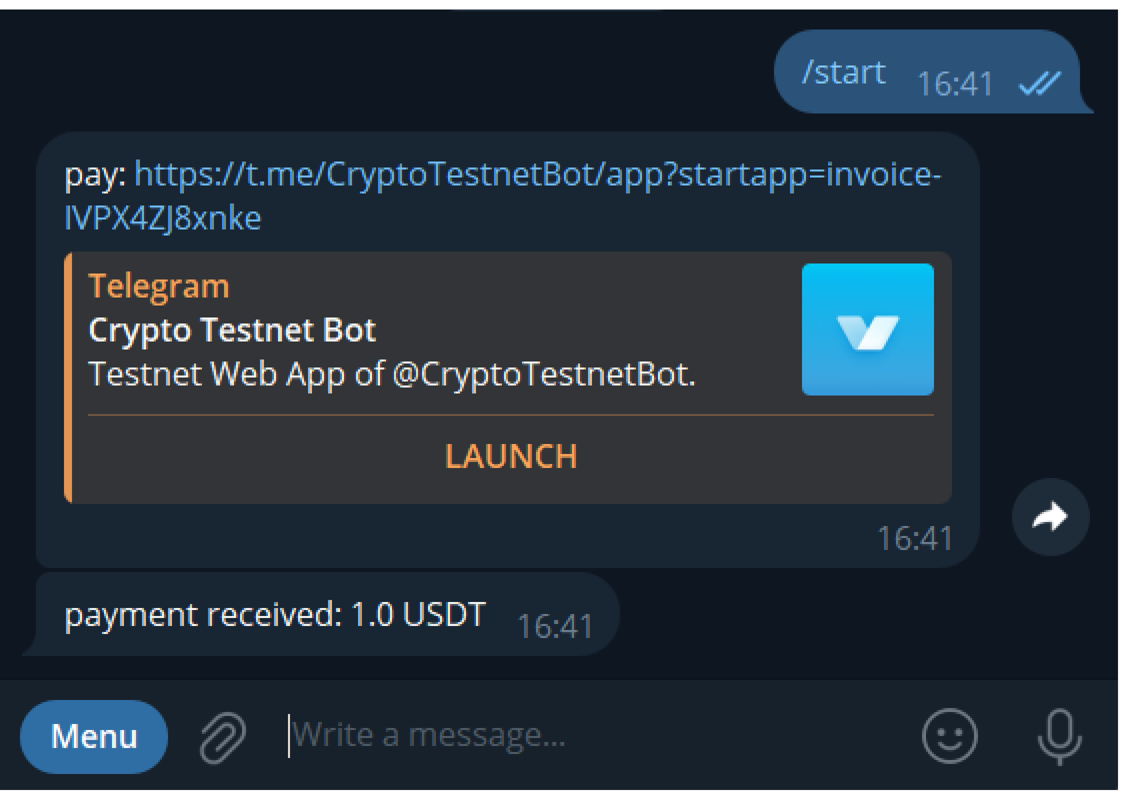
```
from pyrogram import Client, filters
from pyrogram.types import Message
from aiosend import CryptoPay
from aiosend.types import Invoice

app = Client("my_account")
cp = CryptoPay("TOKEN")

@app.on_message(filters.private)
async def get_invoice(client: Client, message: Message) -> None:
    invoice = await cp.create_invoice(1, "USDT")
    await message.reply_text(
        f"pay: {invoice.mini_app_invoice_url}",
    )
    invoice.poll(message=message)

@cp.invoice_paid()
async def handle_payment(
    invoice: Invoice,
    message: Message,
) -> None:
    await message.reply_text(
```

(continues on next page)

(continued from previous page)

```
        f"payment received: {invoice.amount} {invoice.asset}",
    )

cp.start_polling(app.start)
```

Preview



3.5.7 Telethon

Usage example with Telethon

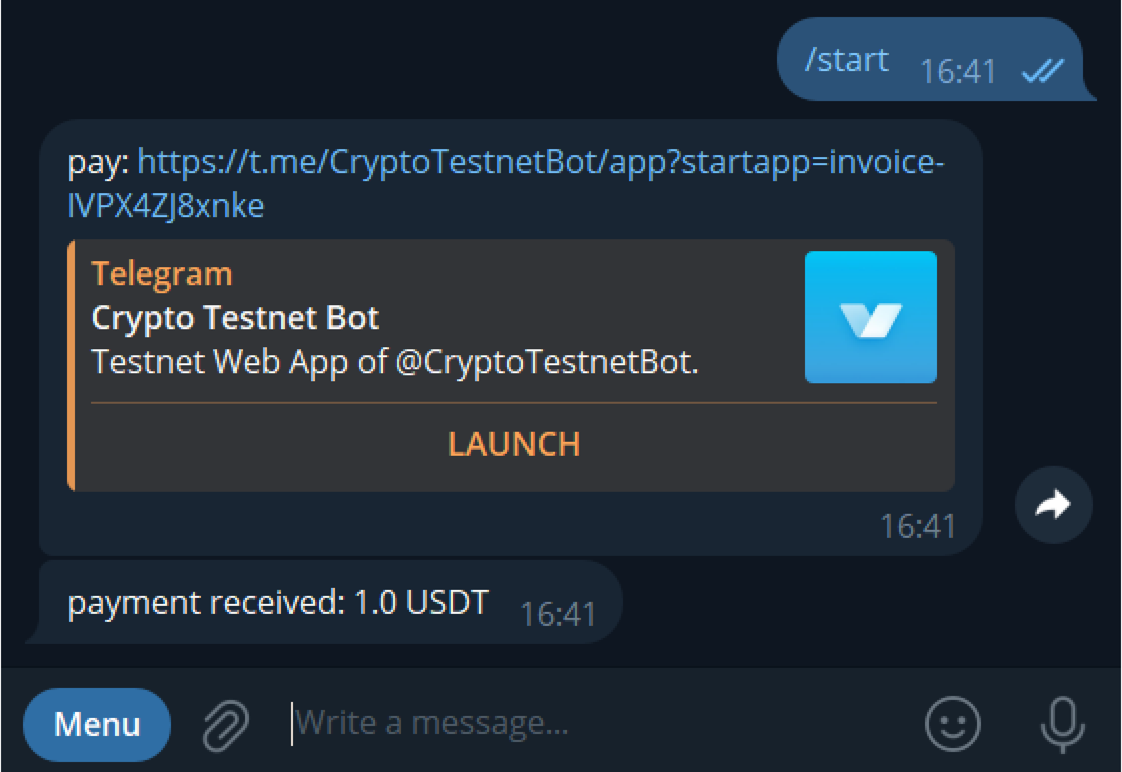
```
from telethon import TelegramClient, events
from telethon.events.newmessage import NewMessage
from aiosend import CryptoPay
from aiosend.types import Invoice

api_id = 12345678
api_hash = "API HASH"

cp = CryptoPay("TOKEN")
client = TelegramClient("anon", api_id, api_hash)

@client.on(events.NewMessage(func=lambda e: e.is_private))
async def get_invoice(message: NewMessage.Event) -> None:
    invoice = await cp.create_invoice(1, "USDT")
    await client.send_message(
```

(continues on next page)

(continued from previous page)

```
        message.chat_id,
        f"pay: {invoice.mini_app_invoice_url}",
    )
    invoice.poll(chat_id=message.chat_id)

@cp.invoice_paid()
async def handle_payment(
    invoice: Invoice,
    chat_id: int,
) -> None:
    await client.send_message(
        chat_id,
        f"payment received: {invoice.amount} {invoice.asset}",
    )

client.start()
cp.start_polling()
```

Preview



PYTHON MODULE INDEX

a

aiosend.client.network, 22
aiosend.client.session, 30
aiosend.enums, 19
aiosend.exceptions, 23
aiosend.types, 13

INDEX

A

accepted_assets (*aiosend.types.Invoice attribute*), 15
ACTIVATED (*aiosend.enums.CheckStatus attribute*), 20
activated_at (*aiosend.types.Check attribute*), 14
ACTIVE (*aiosend.enums.CheckStatus attribute*), 20
ACTIVE (*aiosend.enums.InvoiceStatus attribute*), 21
AED (*aiosend.enums.Fiat attribute*), 20
AiohttpManager (*class in aiosend.webhook*), 34
AiohttpSession (*class in aiosend.client.session*), 30
aiosend.client.network
    module, 22
aiosend.client.session
    module, 30
aiosend.enums
    module, 19
aiosend.exceptions
    module, 23
aiosend.types
    module, 13
allow_anonymous (*aiosend.types.Invoice attribute*), 15
allow_comments (*aiosend.types.Invoice attribute*), 15
AMD (*aiosend.enums.Fiat attribute*), 20
amount (*aiosend.types.Check attribute*), 14
amount (*aiosend.types.Invoice attribute*), 15
amount (*aiosend.types.Transfer attribute*), 18
APIError, 23
APITimeoutError, 23
App (*class in aiosend.types*), 13
app_id (*aiosend.types.App attribute*), 13
AppStats (*class in aiosend.types*), 13
asset (*aiosend.types.Check attribute*), 14
asset (*aiosend.types.Invoice attribute*), 15
asset (*aiosend.types.Transfer attribute*), 18
Asset (*class in aiosend.enums*), 19
available (*aiosend.types.Balance attribute*), 13
AZN (*aiosend.enums.Fiat attribute*), 20

B

Balance (*class in aiosend.types*), 13
base (*aiosend.client.network.Network attribute*), 22
BaseRouter (*class in aiosend._events*), 30
BaseSession (*class in aiosend.client.session*), 30
BNB (*aiosend.enums.Asset attribute*), 19
BONK (*aiosend.enums.Asset attribute*), 19
bot_check_url (*aiosend.types.Check attribute*), 14
bot_invoice_url (*aiosend.types.Invoice attribute*), 16
BRL (*aiosend.enums.Fiat attribute*), 20
BTC (*aiosend.enums.Asset attribute*), 19
BYN (*aiosend.enums.Fiat attribute*), 20

C

CALLBACK (*aiosend.enums.PaidBtnName attribute*), 21
CATI (*aiosend.enums.Asset attribute*), 19
Check (*class in aiosend.types*), 14
check_id (*aiosend.types.Check attribute*), 14
CheckStatus (*class in aiosend.enums*), 20
CNY (*aiosend.enums.Fiat attribute*), 20
code (*aiosend.types.Currency attribute*), 14
code (*aiosend.types.Error attribute*), 15
comment (*aiosend.types.Invoice attribute*), 16
comment (*aiosend.types.Transfer attribute*), 18
completed_at (*aiosend.types.Transfer attribute*), 18
conversion (*aiosend.types.AppStats attribute*), 13
create_check() (*aiosend.CryptoPay method*), 9
create_invoice() (*aiosend.CryptoPay method*), 8
created_at (*aiosend.types.Check attribute*), 14
created_at (*aiosend.types.Invoice attribute*), 16
created_invoice_count (*aiosend.types.AppStats attribute*), 13
CRYPTO (*aiosend.enums.CurrencyType attribute*), 20
CryptoPayError, 23
CryptoPayObject (*class in aiosend.types*), 14
Currency (*class in aiosend.types*), 14
currency_code (*aiosend.types.Balance attribute*), 14
currency_type (*aiosend.types.Invoice attribute*), 16
CurrencyType (*class in aiosend.enums*), 20

D

decimals (*aiosend.types.Currency attribute*), 14
delay (*aiosend.polling.PollingConfig attribute*), 39
delete() (*aiosend.types.Check method*), 26
delete() (*aiosend.types.Invoice method*), 26
delete_all_checks() (*aiosend.CryptoPay method*), 25

delete_all_invoices() (*aiosend.CryptoPay method*), 25
delete_check() (*aiosend.CryptoPay method*), 10
delete_invoice() (*aiosend.CryptoPay method*), 9
description (*aiosend.types.Invoice attribute*), 16
DeserializationError, 23
DOGS (*aiosend.enums.Asset attribute*), 19

E

end_at (*aiosend.types.AppStats attribute*), 13
error (*aiosend.types.Response attribute*), 18
Error (*class in aiosend.types*), 15
ETH (*aiosend.enums.Asset attribute*), 19
EUR (*aiosend.enums.Fiat attribute*), 20
EventObserver (*class in aiosend._events*), 31
exchange() (*aiosend.CryptoPay method*), 23
ExchangeRate (*class in aiosend.types*), 15
expiration_date (*aiosend.types.Invoice attribute*), 16
EXPIRED (*aiosend.enums.InvoiceStatus attribute*), 21

F

FastAPIManager (*class in aiosend.webhook*), 34
fee (*aiosend.types.Invoice attribute*), 16
fee_amount (*aiosend.types.Invoice attribute*), 16
fee_asset (*aiosend.types.Invoice attribute*), 16
fee_in_usd (*aiosend.types.Invoice attribute*), 16
FIAT (*aiosend.enums.CurrencyType attribute*), 20
fiat (*aiosend.types.Invoice attribute*), 16
Fiat (*class in aiosend.enums*), 20
filter() (*aiosend.PayloadData class method*), 40
FlaskManager (*class in aiosend.webhook*), 35

G

GBP (*aiosend.enums.Fiat attribute*), 20
GEL (*aiosend.enums.Fiat attribute*), 20
get_balance() (*aiosend.CryptoPay method*), 12
get_balance_by_asset() (*aiosend.CryptoPay method*), 24
get_check() (*aiosend.CryptoPay method*), 25
get_check_image() (*aiosend.client.network.Network method*), 22
get_checks() (*aiosend.CryptoPay method*), 11
get_checks() (*aiosend.polling.PollingManager method*), 40
get_currencies() (*aiosend.CryptoPay method*), 12
get_exchange_rates() (*aiosend.CryptoPay method*), 12
get_image() (*aiosend.types.Check method*), 27
get_invoice() (*aiosend.CryptoPay method*), 24
get_invoices() (*aiosend.CryptoPay method*), 11
get_invoices() (*aiosend.polling.PollingManager method*), 40
get_me() (*aiosend.CryptoPay method*), 8
get_qr() (*aiosend.client.network.Network method*), 22
get_rates_image() (*aiosend.client.network.Network method*), 22
get_rates_image() (*aiosend.CryptoPay method*), 25
get_stats() (*aiosend.CryptoPay method*), 12
get_transfers() (*aiosend.CryptoPay method*), 11
GRAM (*aiosend.enums.Asset attribute*), 19

H

hash (*aiosend.types.Check attribute*), 14
hash (*aiosend.types.Invoice attribute*), 16
hidden_message (*aiosend.types.Invoice attribute*), 16
HMSTR (*aiosend.enums.Asset attribute*), 19

I

IDR (*aiosend.enums.Fiat attribute*), 20
ILS (*aiosend.enums.Fiat attribute*), 20
include_router() (*aiosend._events.BaseRouter method*), 31
include_routers() (*aiosend._events.BaseRouter method*), 31
INR (*aiosend.enums.Fiat attribute*), 20
Invoice (*class in aiosend.types*), 15
invoice_id (*aiosend.types.Invoice attribute*), 16
INVOICE_PAID (*aiosend.enums.UpdateType attribute*), 21
InvoiceStatus (*class in aiosend.enums*), 21
is_blockchain (*aiosend.types.Currency attribute*), 14
is_crypto (*aiosend.types.ExchangeRate attribute*), 15
is_fiat (*aiosend.types.Currency attribute*), 14
is_fiat (*aiosend.types.ExchangeRate attribute*), 15
is_stablecoin (*aiosend.types.Currency attribute*), 14
is_swapped (*aiosend.types.Invoice attribute*), 16
is_valid (*aiosend.types.ExchangeRate attribute*), 15
items (*aiosend.types.ItemsList attribute*), 18
ItemsList (*class in aiosend.types*), 18

J

JET (*aiosend.enums.Asset attribute*), 19

K

KGS (*aiosend.enums.Fiat attribute*), 20
KZT (*aiosend.enums.Fiat attribute*), 20

L

LTC (*aiosend.enums.Asset attribute*), 19

M

MAINNET (*in module aiosend*), 21
MAJOR (*aiosend.enums.Asset attribute*), 19
MELANIA (*aiosend.enums.Asset attribute*), 19
MEMHASH (*aiosend.enums.Asset attribute*), 19
MethodValuesError, 23

mini_app_invoice_url (*aiosend.types.Invoice attribute*), 16
module
    aiosend.client.network, 22
    aiosend.client.session, 30
    aiosend.enums, 19
    aiosend.exceptions, 23
    aiosend.types, 13
MY (*aiosend.enums.Asset attribute*), 19

N

name (*aiosend.client.network.Network attribute*), 22
name (*aiosend.types.App attribute*), 13
name (*aiosend.types.Currency attribute*), 15
name (*aiosend.types.Error attribute*), 15
Network (*class in aiosend.client.network*), 22
NOT (*aiosend.enums.Asset attribute*), 19

O

ok (*aiosend.types.Response attribute*), 18
onhold (*aiosend.types.Balance attribute*), 14
OPENBOT (*aiosend.enums.PaidBtnName attribute*), 21
OPENCHANNEL (*aiosend.enums.PaidBtnName attribute*), 21

P

pack() (*aiosend.PayloadData method*), 40
PAID (*aiosend.enums.InvoiceStatus attribute*), 21
paid_amount (*aiosend.types.Invoice attribute*), 16
paid_anonymously (*aiosend.types.Invoice attribute*), 17
paid_asset (*aiosend.types.Invoice attribute*), 17
paid_at (*aiosend.types.Invoice attribute*), 17
paid_btn_name (*aiosend.types.Invoice attribute*), 17
paid_btn_url (*aiosend.types.Invoice attribute*), 17
paid_fiat_rate (*aiosend.types.Invoice attribute*), 17
paid_invoice_count (*aiosend.types.AppStats attribute*), 13
paid_usd_rate (*aiosend.types.Invoice attribute*), 17
PaidBtnName (*class in aiosend.enums*), 21
pay_url (*aiosend.types.Invoice attribute*), 17
payload (*aiosend.types.Invoice attribute*), 17
payload (*aiosend.types.Update attribute*), 18
PayloadData (*class in aiosend*), 40
payment_processing_bot_username (*aiosend.types.App attribute*), 13
PEPE (*aiosend.enums.Asset attribute*), 19
PLN (*aiosend.enums.Fiat attribute*), 21
poll() (*aiosend.types.Check method*), 27
poll() (*aiosend.types.Invoice method*), 27
PollingConfig (*class in aiosend.polling*), 39
PollingManager (*class in aiosend.polling*), 40
PollingRouter (*class in aiosend.polling*), 31

Q

qr (*aiosend.types.Check property*), 26
qr (*aiosend.types.Invoice property*), 26

R

rate (*aiosend.types.ExchangeRate attribute*), 15
register() (*aiosend._events.EventObserver method*), 31
register_handler() (*aiosend.webhook.AiohttpManager method*), 34
register_handler() (*aiosend.webhook.FastAPIManager method*), 35
register_handler() (*aiosend.webhook.FlaskManager method*), 35
register_handler() (*aiosend.webhook.WebhookManager method*), 34
request() (*aiosend.client.session.AiohttpSession method*), 30
request() (*aiosend.client.session.BaseSession method*), 30
request_date (*aiosend.types.Update attribute*), 19
Response (*class in aiosend.types*), 18
result (*aiosend.types.Response attribute*), 18
RUB (*aiosend.enums.Fiat attribute*), 21

S

SEND (*aiosend.enums.Asset attribute*), 19
SOL (*aiosend.enums.Asset attribute*), 19
source (*aiosend.types.ExchangeRate attribute*), 15
spend_id (*aiosend.types.Transfer attribute*), 18
start_at (*aiosend.types.AppStats attribute*), 13
start_polling() (*aiosend.polling.PollingManager method*), 40
status (*aiosend.types.Check attribute*), 14
status (*aiosend.types.Invoice attribute*), 17
status (*aiosend.types.Transfer attribute*), 18
swap_to (*aiosend.types.Invoice attribute*), 17
swapped_output (*aiosend.types.Invoice attribute*), 17
swapped_rate (*aiosend.types.Invoice attribute*), 17
swapped_to (*aiosend.types.Invoice attribute*), 17
swapped_uid (*aiosend.types.Invoice attribute*), 17
swapped_usd_amount (*aiosend.types.Invoice attribute*), 17
swapped_usd_rate (*aiosend.types.Invoice attribute*), 17

T

target (*aiosend.types.ExchangeRate attribute*), 15
TESTNET (*in module aiosend*), 21
THB (*aiosend.enums.Fiat attribute*), 21
timeout (*aiosend.polling.PollingConfig attribute*), 39
TJS (*aiosend.enums.Fiat attribute*), 21
TON (*aiosend.enums.Asset attribute*), 19
Transfer (*class in aiosend.types*), 18

transfer() (*aiosend.CryptoPay method*), 10
transfer_id (*aiosend.types.Transfer attribute*), 18
trigger() (*aiosend._events.EventObserver method*), 31
TRUMP (*aiosend.enums.Asset attribute*), 19
TRX (*aiosend.enums.Asset attribute*), 20
TRY (*aiosend.enums.Fiat attribute*), 21

U

UAH (*aiosend.enums.Fiat attribute*), 21
unique_users_count (*aiosend.types.AppStats attribute*), 13
unpack() (*aiosend.PayloadData class method*), 40
Update (*class in aiosend.types*), 18
update() (*aiosend.types.Balance method*), 25
update() (*aiosend.types.Check method*), 25
update() (*aiosend.types.ExchangeRate method*), 26
update() (*aiosend.types.Invoice method*), 26
update_id (*aiosend.types.Update attribute*), 19
update_type (*aiosend.types.Update attribute*), 19
UpdateType (*class in aiosend.enums*), 21
url (*aiosend.types.Currency attribute*), 15
url() (*aiosend.client.network.Network method*), 22
USD (*aiosend.enums.Fiat attribute*), 21
usd_rate (*aiosend.types.Invoice attribute*), 18
USDC (*aiosend.enums.Asset attribute*), 20
USDT (*aiosend.enums.Asset attribute*), 20
user_id (*aiosend.types.Transfer attribute*), 18
UZS (*aiosend.enums.Fiat attribute*), 21

V

VIEWITEM (*aiosend.enums.PaidBtnName attribute*), 21
volume (*aiosend.types.AppStats attribute*), 13

W

web_app_invoice_url (*aiosend.types.Invoice attribute*), 18
WebhookManager (*class in aiosend.webhook*), 34
WebhookRouter (*class in aiosend.webhook*), 31
WIF (*aiosend.enums.Asset attribute*), 20
WrongNetworkError, 23

X

XAUT (*aiosend.enums.Asset attribute*), 20