TellerBot

Aug 02, 2021

Contents:

CHAPTER 1

tellerbot

1.1 src package

1.1.1 Subpackages

src.escrow package

Subpackages

src.escrow.blockchain package

Submodules

src.escrow.blockchain.golos_blockchain module

Module contents

class src.escrow.blockchain.**BaseBlockchain**

Bases: abc.ABC

Abstract class to represent blockchain node client for escrow exchange.

address = None

Address used by bot.

assets = frozenset()

Frozen set of assets supported by blockchain.

check_timeout(*offer_id: bson.objectid.ObjectId*) → None

Start transaction check timeout asynchronously.

Parameters **offer_id** – _id of escrow offer.

check_transaction(*, *offer_id: bson.objectid.ObjectId*, *from_address: str*, *amount_with_fee: decimal.Decimal*, *amount_without_fee: decimal.Decimal*, *asset: str*, *memo: str*, *transaction_time: float*) → bool

Check transaction in history of escrow address.

Parameters

- **offer_id** – `_id` of escrow offer.
- **from_address** – Address which sent assets.
- **amount_with_fee** – Amount of transferred asset with fee added.
- **amount_without_fee** – Amount of transferred asset with fee substracted.
- **asset** – Transferred asset.
- **memo** – Memo in blockchain transaction.
- **transaction_time** – Start of transaction check.

Returns Queue member with timeout handler or None if queue member is timeouted.

close()

Close connection with blockchain node.

connect() → None

Establish connection with blockchain node.

create_queue() → List[Dict[str, Any]]

Create queue from unconfirmed transactions in database.

explorer = '{}'

Template of URL to transaction in blockchain explorer. Should contain `{}` which gets replaced with transaction id.

get_limits(*asset: str*) → src.escrow.blockchain.InsuranceLimits

Get maximum amounts of `asset` which will be insured during escrow exchange.

Escrow offer starts only if sum of it doesn't exceed these limits.

get_min_time(*queue: List[Dict[str, Any]]*) → float

Get timestamp of earliest transaction from `queue`.

is_block_confirmed(*block_num: int*, *op: Mapping[str, Any]*) → bool

Check if block # `block_num` has `op` after confirmation.

Check block on blockchain-specific conditions to consider it confirmed.

Parameters

- **block_num** – Number of block to check.
- **op** – Operation to check.

name = None

Internal name of blockchain referenced in `config.ESCROW_FILENAME`.

nodes

Get list of node URLs.

schedule_timeout(*queue_member: Dict[str, Any]*) → Optional[Dict[str, Any]]

Schedule timeout of transaction check.

transfer(*to: str*, *amount: decimal.Decimal*, *asset: str*, *memo: str = ''*) → str

Transfer `asset` from `self.address`.

Parameters

- **to** – Address assets are transferred to.
- **amount** – Amount of transferred asset.
- **asset** – Transferred asset.

Returns URL to transaction in blockchain explorer.

trx_url(*trx_id: str*) → str
Get URL on transaction with ID `trx_id` on explorer.

wif
Get private key encoded to WIF.

exception src.escrow.blockchain.**BlockchainConnectionError**
Bases: `Exception`

Unsuccessful attempt at connection to blockchain node.

class src.escrow.blockchain.**InsuranceLimits**
Bases: `tuple`

Maximum amount of insured asset.

single
Limit on sum of a single offer.

total
Limit on overall sum of offers.

class src.escrow.blockchain.**StreamBlockchain**
Bases: *src.escrow.blockchain.BaseBlockchain*

Blockchain node client supporting continuous stream to check transaction.

add_to_queue(***kwargs*)
Add transaction to self._queue to be checked.

Same parameters as in `self.check_transaction`.

check_timeout(*offer_id: bson.objectid.ObjectId*) → None
Start transaction check timeout asynchronously.

Parameters **offer_id** – `_id` of escrow offer.

remove_from_queue(*offer_id: bson.objectid.ObjectId*) → Optional[Mapping[str, Any]]
Remove transaction with specified `offer_id` value from `self._queue`.

Parameters **offer_id** – `_id` of escrow offer.

Returns True if transaction was found and False otherwise.

start_streaming() → None
Start streaming in background asynchronous task.

stream() → None
Stream new blocks and check if they contain transactions from `self._queue`.

Use built-in method to subscribe to new blocks if node has it, otherwise get new blocks in blockchain-specific time interval between blocks.

If block contains desired transaction, call `self._confirmation_callback`. If it returns True, remove transaction from `self._queue` and stop streaming if `self._queue` is empty.

exception src.escrow.blockchain.**TransferError**
Bases: Exception

Unsuccessful attempt at transfer.

Submodules

src.escrow.escrow_offer module

class src.escrow.escrow_offer.**EscrowOffer**(*_id: bson.objectid.ObjectId, order: bson.objectid.ObjectId, buy: str, sell: str, type: str, escrow: str, time: float, init: Mapping[str, Any], counter: Mapping[str, Any], pending_input_from: Optional[int] = None, sum_currency: Optional[str] = None, sum_buy: Optional[bson.decimal128.Decimal128] = None, sum_sell: Optional[bson.decimal128.Decimal128] = None, sum_fee_up: Optional[bson.decimal128.Decimal128] = None, sum_fee_down: Optional[bson.decimal128.Decimal128] = None, insured: Optional[bson.decimal128.Decimal128] = None, react_time: Optional[float] = None, transaction_time: Optional[float] = None, cancel_time: Optional[float] = None, bank: Optional[str] = None, memo: Optional[str] = None, trx_id: Optional[str] = None, unsent: Optional[bool] = None*)

Bases: object

Class used to represent escrow offer.

Attributes correspond to fields in database document.

bank = None
Bank of fiat currency.

buy = None
Currency which order creator wants to buy.

cancel_time = None
Unix time stamp of offer cancellation.

counter = None
Object representing counteragent of escrow.

delete_document() → None
Archive and delete corresponding document in database.

escrow = None
Currency which is held during exchange.

init = None
Object representing initiator of escrow.

insert_document() → None
Convert self to document and insert to database.

insured = None
Amount of insured currency.

memo = None
Required memo in blockchain transaction.

order = None
Primary key value of corresponding order document.

pending_input_from = None
Telegram ID of user required to send message to bot.

react_time = None
Unix time stamp of counteragent first reaction to sent offer.

sell = None
Currency which order creator wants to sell.

sum_buy = None
Amount in `buy` currency.

sum_currency = None
Temporary field of currency in which user is sending amount.

sum_fee_down = None
Amount of held currency with agreed fee substracted.

sum_fee_up = None
Amount of held currency with agreed fee added.

sum_sell = None
Amount in `sell` currency.

time = None
Unix time stamp of offer creation.

transaction_time = None
Unix time stamp since which transaction should be checked.

trx_id = None
ID of verified transaction.

type = None
Type of offer. Field of currency which is held during exchange.

unsent = None
True if non-escrow token sender hasn't confirmed their transfer.

update_document(*update*) → None
Update corresponding document in database.

Parameters **update** – Document with update operators or aggregation pipeline sent to MongoDB.

src.escrow.escrow_offer.**asdict**(*instance*)
Represent class instance as dictionary excluding None values.

Module contents

src.escrow.**close_blockchains**()
Run close() method on every blockchain instance.

src.escrow.**connect_to_blockchains**()
Run connect() method on every blockchain instance.

src.escrow.**get_escrow_instance**(*asset: str*)
Find blockchain instance which supports asset.

src.handlers package

Submodules

src.handlers.base module

src.handlers.base.**inline_control_buttons**(*back: bool = True*, *skip: bool = True*, *cancel: bool = True*) → List[aiogram.types.inline_keyboard.InlineKeyboardButton]
Create inline button row with translated labels to control current state.

src.handlers.base.**orders_list**(*cursor: pymongo.cursor.Cursor*, *chat_id: int*, *start: int*, *quantity: int*, *buttons_data: str*, *user_id: Optional[int] = None*, *message_id: Optional[int] = None*, *invert: Optional[bool] = None*) → None
Send list of orders.

Parameters

- **cursor** – Cursor of MongoDB query to orders.
- **chat_id** – Telegram ID of current chat.
- **start** – Start index.
- **quantity** – Quantity of orders in cursor.
- **buttons_data** – Beginning of callback data of left/right buttons.
- **user_id** – Telegram ID of current user if cursor is not user-specific.
- **message_id** – Telegram ID of message to edit.
- **invert** – Invert all prices.

src.handlers.base.**show_order**(*order: Mapping[str, Any]*, *chat_id: int*, *user_id: int*, *message_id: Optional[int] = None*, *location_message_id: Optional[int] = None*, *show_id: bool = False*, *invert: Optional[bool] = None*, *edit: bool = False*, *locale: Optional[str] = None*)
Send detailed order.

Parameters

- **order** – Order document.
- **chat_id** – Telegram ID of chat to send message to.
- **user_id** – Telegram user ID of message receiver.
- **message_id** – Telegram ID of message to edit.

- **location_message_id** – Telegram ID of message with location object. It is deleted when **Hide** inline button is pressed.
- **show_id** – Add ID of order to the top.
- **invert** – Invert price.
- **edit** – Enter edit mode.
- **locale** – Locale of message receiver.

src.handlers.base.**start_keyboard**() → aiogram.types.reply_keyboard.ReplyKeyboardMarkup
Create reply keyboard with main menu.

src.handlers.creation module

Handlers for order creation.

Handlers decorated with `state_handler` are called by `change_state` when user skips (where it is possible) or goes back to corresponding step using back/skip inline buttons. Handlers decorated with `private_handler` are called when user sends value.

src.handlers.creation.**cancel_button**(*call: aiogram.types.callback_query.CallbackQuery*, *state: aiogram.dispatcher.storage.FSMContext*)
React to cancel button.

src.handlers.creation.**cancel_order_creation**(*user_id: int*, *chat_id: int*)
Cancel order creation.

src.handlers.creation.**change_state**(*call: aiogram.types.callback_query.CallbackQuery*, *state: aiogram.dispatcher.storage.FSMContext*)
React to back/skip button query.

src.handlers.creation.**choose_buy**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set currency user wants to buy and ask for one they want to sell.

src.handlers.creation.**choose_buy_gateway**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set gateway of buy currency and ask for sell currency.

src.handlers.creation.**choose_comments**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set comments and finish order creation.

src.handlers.creation.**choose_comments_handler**(*call: aiogram.types.callback_query.CallbackQuery*)
Finish order creation.

src.handlers.creation.**choose_duration**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set duration and ask for comments.

src.handlers.creation.**choose_location**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set location from Telegram object and ask for duration.

src.handlers.creation.**choose_payment_system**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set payment system and ask for location.

src.handlers.creation.**choose_price**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set price and ask for sum currency.

src.handlers.creation.**choose_sell**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set currency user wants to sell and ask for price.

src.handlers.creation.**choose_sell_gateway**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set gateway of sell currency and ask for price.

src.handlers.creation.**choose_sum**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Set sum.

If price and sum in another currency were not specified, ask for sum in another currency. Otherwise calculate it if price was specified, and, finally, ask for cashless payment system.

src.handlers.creation.**choose_sum_currency**(*call: aiogram.types.callback_query.CallbackQuery*)
Set sum currency and ask for sum in that currency.

src.handlers.creation.**comment_handler**(*call: aiogram.types.callback_query.CallbackQuery*)
Ask for comments.

src.handlers.creation.**duration_handler**(*call: aiogram.types.callback_query.CallbackQuery*)
Ask for duration.

src.handlers.creation.**geocoded_location**(*call: aiogram.types.callback_query.CallbackQuery*)
Choose location from list of options and ask for duration.

src.handlers.creation.**get_currency_with_gateway**(*currency_type: str*, *message: aiogram.types.message.Message*)
Try to append gateway from message text to currency.

src.handlers.creation.**invert_price**(*call: aiogram.types.callback_query.CallbackQuery*)
Change currency of price.

src.handlers.creation.**location_handler**(*call: aiogram.types.callback_query.CallbackQuery*)
Ask for location.

src.handlers.creation.**match_currency**(*currency_type: str*, *message: aiogram.types.message.Message*)
Match message text with currency pattern.

src.handlers.creation.**payment_system_handler**(*call: aiogram.types.callback_query.CallbackQuery*)
Ask for cashless payment system.

src.handlers.creation.**price_ask**(*call: aiogram.types.callback_query.CallbackQuery, order: Mapping[str, Any], price_currency: str*)
Edit currency of price in message to `price_currency` field value.

src.handlers.creation.**price_handler**(*call: aiogram.types.callback_query.CallbackQuery*)
Ask for price.

src.handlers.creation.**set_order**(*order: MutableMapping[str, Any], chat_id: int*)
Set missing values and finish order creation.

src.handlers.creation.**set_price_state**(*message: aiogram.types.message.Message, order: Mapping[str, Any]*)
Ask for price.

src.handlers.creation.**sum_handler**(*call: aiogram.types.callback_query.CallbackQuery*)
Ask for sum currency.

src.handlers.creation.**text_location**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Find location by name.

If there is only one option, set it and ask for duration. Otherwise send a list of these options for user to choose.

src.handlers.creation.**whitelisting_request**(*call: aiogram.types.callback_query.CallbackQuery*)
Send whitelisting request to support or increment requests count.

src.handlers.escrow module

Handlers for escrow exchange.

src.handlers.escrow.**accept_insurance**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)
Ask for fee payment agreement after accepting partial insurance.

src.handlers.escrow.**accept_offer**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)
React to counteragent accepting offer by asking for fee payment agreement.

src.handlers.escrow.**add_cashback**(*currency*, *amount*, *sum_fee_up*, *sum_fee_down*, *sender_user*, *recipient_user*)
Create cashback documents from escrow exchange.

src.handlers.escrow.**ask_credentials**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)
Update offer with `update_dict` and start asking transfer information.

Ask to choose bank if user is initiator and there is a fiat currency. Otherwise ask receive address.

src.handlers.escrow.**ask_fee**(*user_id: int*, *chat_id: int*, *offer: src.escrow.escrow_offer.EscrowOffer*)
Ask fee of any party.

src.handlers.escrow.**call_later**(*delay: float*, *callback: Callable*, **args*, ***kwargs*)
Call `callback(*args, **kwargs)` asynchronously after `delay` seconds.

src.handlers.escrow.**cancel_offer**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)
React to offer cancellation.

While first party is transferring, second party can't cancel offer, because we can't be sure that first party hasn't already completed transfer before confirming.

src.handlers.escrow.**check_transaction**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)
Start transaction check.

src.handlers.escrow.**choose_bank**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)
Set chosen bank and continue.

Because bank is chosen by initiator, ask for receive address if they receive escrow asset.

src.handlers.escrow.**complete_offer**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)
Release escrow asset and finish exchange.

src.handlers.escrow.**create_memo**(*offer: src.escrow.escrow_offer.EscrowOffer*, ***, *transfer: bool*, *counter_send_address: Optional[str] = None*)
Create memo for transfer.

src.handlers.escrow.**decline_fee**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)
Decline fee and start asking transfer information.

src.handlers.escrow.**decline_offer**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)

React to counteragent declining offer.

src.handlers.escrow.**edit_keyboard**(*offer_id: bson.objectid.ObjectId*, *chat_id: int*, *message_id: int*, *keyboard: aiogram.types.inline_keyboard.InlineKeyboardMarkup*)

Edit inline keyboard markup of message.

Parameters

- **offer_id** – Primary key value of offer document connected with message.
- **chat_id** – Telegram chat ID of message.
- **message_id** – Telegram ID of message.
- **keyboard** – New inline keyboard markup.

src.handlers.escrow.**escrow_callback_handler**(**args*, *state=<State '*'>*, ***kwargs*)

Simplify handling callback queries during escrow exchange.

Add offer of `EscrowOffer` to arguments of decorated callback query handler.

src.handlers.escrow.**escrow_message_handler**(**args*, ***kwargs*)

Simplify handling messages during escrow exchange.

Add offer of `EscrowOffer` to arguments of decorated private message handler.

src.handlers.escrow.**final_offer_confirmation**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Ask not escrow asset receiver to confirm transfer.

src.handlers.escrow.**full_card_number_message**(*message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

React to sent message while sending full card number to fiat sender.

src.handlers.escrow.**full_card_number_request**(*chat_id: int*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Ask to send full card number.

src.handlers.escrow.**full_card_number_sent**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Confirm that full card number is sent and ask for first and last 4 digits.

src.handlers.escrow.**get_card_number**(*text: str*, *chat_id: int*) → Optional[Tuple[str, str]]

Parse first and last 4 digits from card number in `text`.

If parsing is unsuccessful, send warning to `chat_id` and return None. Otherwise return tuple of first and last 4 digits of card number.

src.handlers.escrow.**get_insurance**(*offer: src.escrow.escrow_offer.EscrowOffer*) → decimal.Decimal

Get insurance of escrow asset in `offer` taking limits into account.

src.handlers.escrow.**init_cancel**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Cancel offer on initiator's request.

src.handlers.escrow.**pay_fee**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Accept fee and start asking transfer information.

src.handlers.escrow.**set_counter_send_address**(*address: str*, *message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Set `address` as sender's address of counteragent.

Ask for escrow asset transfer.

src.handlers.escrow.**set_escrow_sum**(*message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Set sum and ask for fee payment agreement.

src.handlers.escrow.**set_init_send_address**(*address: str*, *message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Set `address` as sender's address of initiator.

Send offer to counteragent.

src.handlers.escrow.**set_name**(*message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Set fiat sender's name on card and ask for first and last 4 digits.

src.handlers.escrow.**set_receive_address**(*message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Set escrow asset receiver's address and ask for sender's information.

If there is a fiat currency, which is indicated by existing `bank` field, and user is a fiat sender, ask their name on card. Otherwise ask escrow asset sender's address.

src.handlers.escrow.**set_receive_card_number**(*message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Create address from first and last 4 digits of card number and ask send address.

First and last 4 digits of card number are sent by fiat receiver, so their send address is escrow asset address.

src.handlers.escrow.**set_send_address**(*message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Set send address of any party.

src.handlers.escrow.**set_send_card_number**(*message: aiogram.types.message.Message*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Set first and last 4 digits of any party.

src.handlers.escrow.**validate_offer**(*call: aiogram.types.callback_query.CallbackQuery*, *offer: src.escrow.escrow_offer.EscrowOffer*)

Ask support for manual verification of exchange.

src.handlers.order module

Handlers for showing orders and reacting to query buttons attached to them.

src.handlers.order.**aggregate_orders**(*buy: str*, *sell: str*) → Tuple[motor.core.AgnosticBaseCursor, int]

Aggregate and query orders with specified currency pair.

Return cursor with unexpired orders sorted by price and creation time and quantity of documents in it.

src.handlers.order.**archive_button**(*call: aiogram.types.callback_query.CallbackQuery*, *order: Mapping[str, Any]*)

React to "Archive" or "Unarchive" button by flipping archived flag.

src.handlers.order.**confirm_delete_button**(*call: aiogram.types.callback_query.CallbackQuery*)

Delete order after confirmation button query.

src.handlers.order.**default_duration**(*call: aiogram.types.callback_query.CallbackQuery*, *state: aiogram.dispatcher.storage.FSMContext*)

Repeat default duration.

src.handlers.order.**delete_button**(*call: aiogram.types.callback_query.CallbackQuery*, *order: Mapping[str, Any]*)

React to "Delete" button by asking user to confirm deletion.

src.handlers.order.**edit_button**(*call: aiogram.types.callback_query.CallbackQuery*)

React to "Edit" button by entering edit mode on order.

src.handlers.order.**edit_field**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)

Ask new value of chosen order's field during editing.

src.handlers.order.**escrow_button**(*call: aiogram.types.callback_query.CallbackQuery*, *order: Mapping[str, Any]*)

React to "Escrow" button by starting escrow exchange.

src.handlers.order.**finish_edit**(*user*, *update_dict*)

Update and show order after editing.

src.handlers.order.**get_order_button**(*call: aiogram.types.callback_query.CallbackQuery*, *order: Mapping[str, Any]*)

Choose order from order book.

src.handlers.order.**get_order_command**(*message: aiogram.types.message.Message*)

Get order from ID.

Order ID is indicated after **/id** or **ID:** in message text.

src.handlers.order.**hide_button**(*call: aiogram.types.callback_query.CallbackQuery*)

React to "Hide" button by deleting messages with location object and order.

src.handlers.order.**invert_button**(*call: aiogram.types.callback_query.CallbackQuery*, *order: Mapping[str, Any]*)

React to invert button query.

src.handlers.order.**match_button**(*call: aiogram.types.callback_query.CallbackQuery*, *order: Mapping[str, Any]*)

React to "Match" button by sending list of matched orders.

Matched orders are ones that have the inverted currency pair.

src.handlers.order.**matched_orders_button**(*call: aiogram.types.callback_query.CallbackQuery*)

React to left/right button query in list of orders matched by currency pair.

src.handlers.order.**my_orders_button**(*call: aiogram.types.callback_query.CallbackQuery*)

React to left/right button query in list of user's orders.

src.handlers.order.**order_handler**(*handler: Callable[[aiogram.types.callback_query.CallbackQuery, Mapping[str, Any]], Any]*)

Simplify handling callback queries attached to order.

Add order of `OrderType` to arguments of `handler`.

src.handlers.order.**orders_button**(*call: aiogram.types.callback_query.CallbackQuery*)

React to left/right button query in order book.

src.handlers.order.**show_orders**(*call: aiogram.types.callback_query.CallbackQuery*, *cursor: motor.core.AgnosticBaseCursor*, *start: int*, *quantity: int*, *buttons_data: str*, *invert: bool*, *user_id: Optional[int] = None*)

Send list of orders.

Parameters

- **cursor** – Cursor of MongoDB query to orders.
- **chat_id** – Telegram chat ID.
- **start** – Start index.
- **quantity** – Quantity of orders in cursor.
- **buttons_data** – Beginning of callback data of left/right buttons.
- **user_id** – If cursor is user-specific, Telegram ID of user who created all orders in cursor.
- **invert** – Invert all prices.

src.handlers.order.**similar_button**(*call: aiogram.types.callback_query.CallbackQuery, order: Mapping[str, Any]*)

React to "Similar" button by sending list of similar orders.

Similar orders are ones that have the same currency pair.

src.handlers.order.**unset_button**(*call: aiogram.types.callback_query.CallbackQuery*, *state: aiogram.dispatcher.storage.FSMContext*)

React to "Unset" button by unsetting the edit field.

src.handlers.start_menu module

Handlers for start menu.

src.handlers.start_menu.**choose_locale**(*message: aiogram.types.message.Message*)

Show list of languages.

src.handlers.start_menu.**claim_cashback**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)

Start cashback claiming process by asking currency.

src.handlers.start_menu.**get_referral_link**(*message: aiogram.types.message.Message*)

Send user's referral link and generate if it doesn't exist.

src.handlers.start_menu.**handle_book**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*, *command: Optional[aiogram.dispatcher.filters.builtin.Command.CommandObj] = None*)

Show order book with specified currency pair.

Currency pair is indicated with one or two space separated arguments after **/book** in message text. If two arguments are sent, then first is the currency order's creator wants to sell and second is the currency order's creator wants to buy. If one argument is sent, then it's any of the currencies in a pair.

Any argument can be replaced with *, which results in searching pairs with any currency in place of the wildcard.

Examples:

Command	Description
/book BTC USD	Show orders that sell BTC and buy USD (BTC → USD)
/book BTC *	Show orders that sell BTC and buy any currency
/book * USD	Show orders that sell any currency and buy USD
/book BTC	Show orders that sell or buy BTC
/book * *	Equivalent to /book

src.handlers.start_menu.**handle_create**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Start order creation by asking user for currency they want to buy.

src.handlers.start_menu.**handle_my_orders**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Show user's orders.

src.handlers.start_menu.**handle_start_command**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Handle /start.

Ask for language if user is new or show menu.

src.handlers.start_menu.**help_command**(*message: aiogram.types.message.Message*)
Handle request to support.

src.handlers.start_menu.**locale_button**(*call: aiogram.types.callback_query.CallbackQuery*)
Choose language from list.

src.handlers.start_menu.**locale_keyboard**()
Get inline keyboard markup with available locales.

src.handlers.start_menu.**search_by_creator**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Search orders by creator.

Creator is indicated with username (with or without @) or user ID after **/creator** or **/c** in message text.

In contrast to usernames and user IDs, names aren't unique and therefore not supported.

src.handlers.start_menu.**subcribe_to_pair**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*, *command: aiogram.dispatcher.filters.builtin.Command.CommandObj*)
Manage subscription to pairs.

Currency pair is indicated with two space separated arguments after **/subscribe** or **/unsubscribe** in message text. First argument is the currency order's creator wants to sell and second is the currency order's creator wants to buy.

Similarly to **/book**, any argument can be replaced with *, which results in subscribing to pairs with any currency in place of the wildcard.

Without arguments commands show list of user's subscriptions.

src.handlers.support module

Handlers for interacting with support.

src.handlers.support.**answer_support_ticket**(*message: aiogram.types.message.Message*)
Answer support ticket.

Speech balloon emoji at the beginning is the mark of support's reply to ticket.

src.handlers.support.**contact_support**(*message: aiogram.types.message.Message*, *state: aiogram.dispatcher.storage.FSMContext*)
Send message to support after request in start manu.

src.handlers.support.**handle_reply**(*message: aiogram.types.message.Message*)
Answer support's reply to ticket.

src.handlers.support.**send_message_to_support**(*message: aiogram.types.message.Message*)
Format message and send it to support.

Envelope emoji at the beginning is the mark of support ticket.

src.handlers.support.**toggle_escrow**(*message: aiogram.types.message.Message*)
Toggle escrow availability.

This command makes creation of new escrow offers unavailable if escrow is enabled, and makes it available if it's disabled.

src.handlers.support.**unhelp_button**(*call: aiogram.types.callback_query.CallbackQuery*, *state: aiogram.dispatcher.storage.FSMContext*)
Cancel request to support.

Module contents

Handler modules with default and fallback handlers.

src.handlers.**default_callback_query**(*call: aiogram.types.callback_query.CallbackQuery*)
React to query which has not passed any previous conditions.

If callback query is not answered, button will stuck in loading as if the bot stopped working until it times out. So unknown buttons are better be answered accordingly.

src.handlers.**default_message**(*message: aiogram.types.message.Message*)
React to message which has not passed any previous conditions.

src.handlers.**errors_handler**(*update: aiogram.types.update.Update*, *exception: Exception*)
Handle exceptions when calling handlers.

Send error notification to special chat and warn user about the error.

1.1.2 Submodules

1.1.3 src.app module

src.app.**main**()
Start bot in webhook mode.

Bot's main entry point.

src.app.**on_startup**(*webhook_path=None*, **args*)
Prepare bot before starting.

Set webhook and run background tasks.

1.1.4 src.bot module

class src.bot.**DispatcherManual**(*bot*, *loop=None*, *storage: Optional[aiogram.dispatcher.storage.BaseStorage] = None*, *run_tasks_by_default: bool = False*, *throttling_rate_limit=0.1*, *no_throttle_error=False*, *filters_factory=None*)
Bases: aiogram.dispatcher.dispatcher.Dispatcher

Dispatcher with user availability in database check.

process_update(*update: aiogram.types.update.Update*)
Process update object with user availability in database check.

If bot doesn't know the user, it pretends they sent /start message.

class src.bot.**IncomingHistoryMiddleware**
Bases: aiogram.dispatcher.middlewares.BaseMiddleware

Middleware for storing incoming history.

trigger(*action*, *args*)
Save incoming data in the database.

class src.bot.**TellerBot**(*token: String*, *loop: Union[asyncio.base_events.BaseEventLoop*, *asyncio.events.AbstractEventLoop*, *None] = None*, *connections_limit: Optional[Integer] = None*, *proxy: Optional[String] = None*, *proxy_auth: Optional[aiohttp.helpers.BasicAuth] = None*, *validate_token: Optional[Boolean] = True*, *parse_mode: Optional[String] = None*, *timeout: Union[Integer*, *Float*, *aiohttp.client.ClientTimeout*, *None] = None*)
Bases: aiogram.bot.bot.Bot

Custom bot class.

request(*method*, *data=None*, **args*, ***kwargs*)
Make a request and save it in the database.

src.bot.**private_handler**(**args*, ***kwargs*)
Register handler only for private message.

src.bot.**setup**()
Set API token from config to bot and setup dispatcher.

src.bot.**state_handler**(*state*)
Associate state with decorated handler.

1.1.5 src.database module

class src.database.**MongoStorage**
Bases: aiogram.dispatcher.storage.BaseStorage

MongoDB asynchronous storage for FSM using motor.

close()
Disconnect from MongoDB.

finish(*user: int*, ***kwargs*)
Finish conversation with user.

get_data(*user: int*, ***kwargs*) → Dict[KT, VT]
Get state data of user with Telegram ID user.

get_state(*user: int*, ***kwargs*) → Optional[str]
Get current state of user with Telegram ID user.

reset_state(*user: int*, *with_data: bool = True*, ***kwargs*)
Reset state for user with Telegram ID user.

set_data(*user: int*, *data: Optional[Dict[KT*, *VT]] = None*, ***kwargs*) → None
Set state data data of user with Telegram ID user.

set_state(*user: int*, *state: Optional[str] = None*, ***kwargs*) → None
Set new state state of user with Telegram ID user.

update_data(*user: int*, *data: Optional[Dict[KT, VT]] = None*, ***kwargs*) → None
Update data of user with Telegram ID `user`.

wait_closed() → None
Do nothing.

Motor client does not use this method.

1.1.6 src.i18n module

class src.i18n.**I18nMiddlewareManual**(*domain*, *path*, *default='en'*)
Bases: `aiogram.contrib.middlewares.i18n.I18nMiddleware`

I18n middleware which gets user locale from database.

find_locales() → Dict[str, gettext.NullTranslations]
Load all compiled locales from path and add default fallbacks.

get_user_locale(*action: str, args: Tuple[Any]*) → Optional[str]
Get user locale by querying collection of users in database.

Return value of `locale` field in user's corresponding document if it exists, otherwise return user's Telegram language if possible.

1.1.7 src.money module

exception src.money.**MoneyValueError**
Bases: `Exception`

Inappropriate money argument value.

src.money.**gateway_currency_regexp**(*currency*)
Return regexp that ignores gateway if it isn't specified.

src.money.**money**(*value*) → decimal.Decimal
Try to return normalized money object constructed from `value`.

src.money.**normalize**(*money: decimal.Decimal*, *exp: decimal.Decimal = Decimal('1E-8')*) → decimal.Decimal
Round `money` to `exp` and strip trailing zeroes.

1.1.8 src.notifications module

src.notifications.**order_notification**(*order: Mapping[str, Any]*)
Notify users about order.

Subscriptions to these notifications are managed with **/subscribe** or **/unsubscribe** commands of `start_menu` handlers.

src.notifications.**run_loop**()
Notify order creators about expired orders in infinite loop.

1.1.9 src.states module

class src.states.**Escrow**
Bases: `aiogram.dispatcher.filters.state.StatesGroup`

States of user during escrow exchange.

States are uncontrollable by users and are only used to determine what action user is required to perform. Because there are two parties in escrow exchange and steps are dependant on which currencies are used, states do not define full steps of exchange.

amount = <State 'Escrow:amount'>
Send sum in any of the currencies.

bank = <State 'Escrow:bank'>
Choose escrow initiator's bank from listed.

fee = <State 'Escrow:fee'>
Agree or disagree to pay fee.

full_card = <State 'Escrow:full_card'>
Send fiat receiver's full card number to fiat sender.

name = <State 'Escrow:name'>
Send fiat sender's name on card. Required to verify fiat transfer.

receive_address = <State 'Escrow:receive_address'>
Send escrow asset receiver's address in blockchain.

receive_card_number = <State 'Escrow:receive_card_number'>
Send first and last 4 digits of fiat receiver's card number.

send_address = <State 'Escrow:send_address'>
Escrow asset sender's address in blockchain.

send_card_number = <State 'Escrow:send_card_number'>
Send first and last 4 digits of fiat sender's card number.

class src.states.**OrderCreation**
Bases: aiogram.dispatcher.filters.state.StatesGroup

Steps of order creation.

States represent values user is required to send. They are in order and skippable unless otherwise specified.

amount = <State 'OrderCreation:amount'>
Sum in any of the currencies.

buy = <State 'OrderCreation:buy'>
Currency user wants to buy (unskippable).

buy_gateway = <State 'OrderCreation:buy_gateway'>
Gateway of buy currency (unskippable).

comments = <State 'OrderCreation:comments'>
Any additional comments.

duration = <State 'OrderCreation:duration'>
Duration in days.

location = <State 'OrderCreation:location'>
Location object or location name.

payment_system = <State 'OrderCreation:payment_system'>
Cashless payment system.

price = <State 'OrderCreation:price'>
Price in one of the currencies.

sell = <State 'OrderCreation:sell'>
Currency user wants to sell (unskippable).

sell_gateway = <State 'OrderCreation:sell_gateway'>
Gateway of sell currency (unskippable).

set_order = <State 'OrderCreation:set_order'>
Finish order creation by skipping comments state.

src.states.**asking_support = <State '@:asking_support'>**
Ask support a question.

src.states.**cashback_address = <State '@:cashback_address'>**
Send new value of chosen order's field during editing.

src.states.**field_editing = <State '@:field_editing'>**
Send new value of chosen order's field during editing.

1.1.10 src.whitelist module

Whitelists for orders.

src.whitelist.**currency_keyboard**(*currency_type: str*) → aiogram.types.reply_keyboard.ReplyKeyboardMarkup
Get keyboard with currencies from whitelists.

src.whitelist.**gateway_keyboard**(*currency: str*, *currency_type: str*) → aiogram.types.reply_keyboard.ReplyKeyboardMarkup
Get keyboard with gateways of `currency` from whitelist.

1.1.11 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W