Documentation for Cardano Implementation in hummingbot Client(for sundaeswap)

Table of Contents

A. Added amm_price_sundaeswap.py

Overview

The amm_price_sundaeswap.py script is a custom strategy for Hummingbot that fetches the price of a specified trading pair on the SundaeSwap decentralized exchange (DEX) using Hummingbot's Gateway API. This strategy periodically retrieves price information based on user-defined parameters and logs the results.

Key Components

1. AmmPriceConfig Class

This class defines the configuration for the script using Pydantic's BaseClientModel.

Fields:

- script_file_name *(str)* – The name of the script file (amm_price_sundaeswap.py).
- connector_chain_network *(str)* – Specifies the AMM, blockchain, and network (e.g., sundaeswap_cardano_preview).
- trading_pair *(str)* – The trading pair to fetch the price for (e.g., SBERRY-ADA).
- side *(str)* – The trade direction, either BUY or SELL.
- order_amount *(Decimal)* – The amount of the asset for which the price is being fetched.

2. AmmPriceSundaeswap Class

This class inherits from ScriptStrategyBase and implements the strategy logic.

Key Methods:

init_markets(cls, config: AmmPriceConfig)

- Initializes the required markets using the specified connector, chain, and network.
- Example: {sundaeswap_cardano_preview: {SBERRY-ADA}}.

__init__(self, connectors, config: AmmPriceConfig)

- Initializes the strategy instance and stores the configuration.

on_tick(self)

- Called periodically to trigger price-fetching tasks.
- Ensures that only one fetch operation runs at a time by setting on_going_task.

async_task(self)

- An asynchronous method that:
    1. Extracts the trading pair and connector-chain-network details.
    2. Determines the trade type (BUY or SELL).
    3. Fetches the price using GatewayHttpClient.
    4. Logs the fetched data including:
        - price: Current market price.
        - expectedAmount: Expected output amount in ADA.
        - rawAmount: Raw amount in lovelace.
        - amount: Trade amount.
    5. Waits for 60 seconds before the next fetch cycle.
    6. Handles any exceptions and resets on_going_task.

Gateway API Integration

The script interacts with Hummingbot's **GatewayHttpClient** to fetch real-time price data.

Data Retrieved:

- price – The current market price for the given trading pair.
- expectedAmount – The estimated amount received after execution.
- rawAmount – The amount represented in blockchain base units (e.g., lovelace).
- amount – The total trade amount.

Error Handling

- If an error occurs during the price fetch operation, the script logs the error message and resets on_going_task to allow subsequent executions.

B. Added amm_trade_sundaeswap.py

Overview

The amm_trade_sundaeswap.py script is a custom trading strategy for Hummingbot that executes a single trade on the **SundaeSwap** decentralized exchange (DEX). The script interacts with Hummingbot's **Gateway API** to:

- Fetch real-time prices,
- Adjust the trade price based on a slippage buffer,
- Execute the trade,
- Retrieve wallet balances before and after the trade, and
- Poll transaction status until confirmation.

This strategy is designed to operate on **Cardano-based networks** and ensures only one trade execution per session.

Key Components

1. AmmTradeConfig Class

This class defines the configuration settings required to execute the trade.

Attributes:

- **script_file_name** *(str)* – Name of the script (amm_trade_sundaeswap.py).
- **connector_chain_network** *(str)* – Specifies the AMM, blockchain, and network (e.g., sundaeswap_cardano_preview).
- **trading_pair** *(str)* – The trading pair for the trade (e.g., SBERRY-ADA).
- **side** *(str)* – Trade direction (BUY or SELL).
- **order_amount** *(Decimal)* – The amount of the **base asset** to be traded.

2. AmmTradeSundaeswap Class

This is the main strategy class, extending ScriptStrategyBase. It ensures that only **one trade execution** is performed per session.

Features:

- Fetches the **current market price**.
- Adjusts the price with a **slippage buffer**.
- Retrieves **wallet balances** before and after trade execution.
- **Executes** the trade.
- **Polls transaction status** until confirmation.

Class Variables:

- **slippage_buffer** *(float)* – A **1% slippage buffer** is applied to the trade price.
- **on_going_task** *(bool)* – Ensures that only one task runs at a time.
- **trade_executed** *(bool)* – Prevents multiple trade executions in a session.

Key Methods

1. init_markets(cls, config: AmmTradeConfig)

- Initializes the required markets.
- Example: {sundaeswap_cardano_preview: {SBERRY-ADA}}.

2. on_tick(self)

- Runs periodically and checks if a trade is needed.
- Ensures only **one** trade execution per session using on_going_task and trade_executed flags.
- Triggers the **asynchronous trade task**.

3. async_task(self)

Executes the trade process in the following steps:

1. **Fetch the current price** of the trading pair using GatewayHttpClient.get_price.
2. **Adjust the price** for slippage:
    - If BUY, increases price by 1%.
    - If SELL, decreases price by 1%.
3. **Retrieve wallet details**:
    - Checks if a wallet is configured for the given chain, network, and connector.

    - If no wallet is found, the script notifies the user and exits.
4. **Fetch the wallet balance** before trade execution.
5. **Execute the trade** using GatewayHttpClient.amm_trade.
6. **Poll the transaction status** using poll_transaction().
7. **Fetch updated wallet balance** after trade execution.
8. **Mark trade as executed** (trade_executed = True), ensuring no further trades occur.

4. get_balance(self, chain, network, address, base, quote)

- Fetches the wallet balances for **base and quote assets**.
- Logs the balances before and after trade execution.

5. poll_transaction(self, chain, network, tx_hash)

- Polls the **transaction status** using the transaction hash (txHash).
- Logs the following:
    - Transaction **status** (confirmed or pending).
    - **Block number** and **block height**.
    - **Transaction fees**.
    - **Validity** of the transaction contract.
- The script logs a **"Fetching transaction, please wait..."** message every 5 seconds.
- Polling occurs every **30 seconds** until confirmation.

Error Handling

- If any error occurs during price fetching, wallet retrieval, trade execution, or transaction polling, the script logs the error.
- If the wallet is **not configured**, the script **exits gracefully** with a notification.
- The polling mechanism **retries transaction checks** in case of temporary failures.

C. Added amm_trade_wait_sundaeswap.py

Overview

This script, amm_trade_wait_sundaeswap.py, is designed to execute a trade on the SundaeSwap decentralized exchange using the Hummingbot framework. Unlike the standard AMM trade script, this version waits for a specific price condition before executing the trade. It interacts with the Hummingbot Gateway to fetch prices, check wallet balances, place trades, and poll transaction statuses.

Classes and Configuration

1. AmmTradeConfig

This class defines the configuration parameters required for the script.

Attributes:

- connector_chain_network: Specifies the AMM connector, blockchain, and network (e.g., sundaeswap_cardano_preview).
- trading_pair: The trading pair (e.g., SBERRY-ADA).
- side: Trade side (BUY or SELL).
- order_amount: The amount of the base asset to trade.
- slippage_buffer: A buffer for slippage adjustment (e.g., 0.01 for 1%).
- price_range: The target price at which the user wants to execute the trade.

2. AmmTradeWaitSundaeswap

This is the main strategy class extending ScriptStrategyBase.

Features:

- Fetches the current market price and applies a slippage buffer.
- Waits until the price reaches the user-specified range before executing a trade.
- Ensures only one trade per session.
- Retrieves wallet balances before and after trade execution.
- Polls transaction status until confirmed.

- Implements a waiting mechanism if the price condition is not met.

Key Methods

1. on_tick()

- Ensures a single trade execution by checking flags (on_going_task and trade_executed).
- Triggers the asynchronous trade execution task.

2. async_task()

Handles the trade execution process in multiple steps:

1. Fetches wallet details and updates balances (once per minute).
2. Fetches the current price for the specified trading pair.
3. Applies slippage buffer to the fetched price.
4. Compares the adjusted price with the user-specified price range.
5. If the price meets the condition, the script:
    - Executes the trade via amm_trade.
    - Polls transaction status until confirmed.
    - Updates wallet balances.
    - Marks trade as executed.
6. If the price condition is not met, the script waits 1 minute before checking again.

3. update_balance()

- Retrieves and logs the wallet balances for the base and quote assets.
- Ensures that balance updates occur at most once per minute.

4. is_trade_allowed(current_price)

- Checks whether the current price meets the user's price condition.
- For BUY trades: Executes when the price is **less than or equal to** the target price.
- For SELL trades: Executes when the price is **greater than or equal to** the target price.

5. poll_transaction()

- Continuously polls the transaction status using the transaction hash.
- Logs updates, including whether the trade is confirmed.
- Displays periodic status messages while waiting for transaction confirmation.

D. Added liquidity_bot_sundaeswap.py

This script, liquidity_bot_sundaeswap.py, is designed for managing liquidity on the **SundaeSwap** decentralized exchange (DEX) using the **Hummingbot framework**. It enables users to **add** or **remove** liquidity from liquidity pools (LPs) by interacting with the **Hummingbot Gateway**.

Overview

The bot supports:

1. **Adding liquidity** by supplying two tokens in a specified proportion.
2. **Removing liquidity** by withdrawing a percentage of an existing liquidity pool position.
3. **Fetching real-time price information** to determine the appropriate token amounts.
4. **Polling transaction status** until confirmation.

Configuration (liquidityBotConfig)

The liquidityBotConfig class defines the required settings for running the liquidity bot.

Configuration Parameters

Parameter	Description
connector_chain_network	Specifies the connector, blockchain, and network (e.g., sundaeswap_cardano_preview).
trading_pair	The trading pair (e.g., SBERRY-ADA).
side	The liquidity action (ADD or REMOVE).
token0	The first token to add or remove from the liquidity pool (only required for ADD).
token1	The second token to add or remove from the liquidity

Parameter	Description
	pool (only required for ADD).
amount0	The amount of token0 to provide as liquidity (only required for ADD).
decrease_percentage	The percentage of liquidity to remove (only required for REMOVE).

Conditional Parameters:

- **When side = "ADD":** token0, token1, and amount0 are required.
- **When side = "REMOVE":** decrease_percentage is required.

Class: liquidityBotSundaeswap

This is the main strategy class that extends ScriptStrategyBase. It manages the execution of liquidity actions on **SundaeSwap**.

Attributes

- on_going_task: Prevents multiple executions at the same time.
- liquidity_flow_executed: Ensures liquidity is added/removed only once.
- balance_data: Stores wallet balance information.
- tx_hash: Stores the transaction hash for polling.

Key Methods

1. on_tick()

- Ensures the liquidity task runs only once.
- Uses safe_ensure_future() to execute the task asynchronously.

2. async_task()

Handles the core liquidity management logic:

1. **Fetches and updates balances** before performing any action.
2. **Calculates the required token1 amount** based on the liquidity pool price.
3. **Executes the liquidity action** (ADD or REMOVE).
4. **Polls transaction status** until confirmation.
5. **Updates balances** after the transaction is completed.
6. **Marks the process as executed to prevent repetition.**

3. update_balance()

- Fetches wallet balances for the base token (token0), quote token (token1), and liquidity pool token (LP).
- Ensures balances are updated only once per minute.

4. poll_transaction()

- Continuously checks the transaction status.
- Displays periodic status messages while waiting.
- Marks the transaction as **confirmed** once completed.

Liquidity Actions

Adding Liquidity (ADD)

1. **Calculate amount1**:
    - Fetches the current price of token0/token1 from the liquidity pool.
    - Computes amount1 required based on amount0.
2. **Execute the amm_lp_add command**:
    - Supplies token0 and token1 in the appropriate ratio.
    - Uses a **low-fee tier** and predefined price range (0.5 - 1).
    - Submits the transaction.

Removing Liquidity (REMOVE)

1. **Execute the amm_lp_remove command**:
    - Removes a specified percentage of liquidity from the pool.
    - Uses decrease_percentage to determine the amount to remove.
    - Submits the transaction.

Transaction Polling

After submitting a transaction, the bot continuously checks its status:

- **Pending** → The transaction is still processing.
- **Confirmed** → The transaction is successfully completed.

Polling continues every **30 seconds** until the transaction is confirmed.