Executor Contract

Introduction

The Executor contract allows pullPayment contracts to execute their pullPayment. Executor allows only registered pullPayment contracts to transfer the tokens from customer to merchant using the execute() method.

The Executor contract gets the payment tokens from the customer, swaps the tokens to settlement tokens with the help of the quickswap router, transfers the execution fee in PMA tokens to the fee receiver and transfers the settlement tokens to the merchant.

A typical example of execution is payment of 10 PMA to merchants. In this, if 10% is execution fee then execution fee receiver gets 1 PMA and merchant receives 9 PMA.

Contract version

pragma solidity 0.8.0

Contract constructor

The Executor contract`s constructor takes one argument i.e the main registry contract address which is used to access the registry methods.

The prerequisite for Executor contract deployment is -

- Core Registry contract is deployed
- Pullpayment Registry contract is deployed and have added in Core Registry
- PMA token is deployed and added to the Core Registry.
- Dex router and factory are added in the Core Registry.

Constructor Signature:

- constructor(address registryAddress) RegistryHelper(registryAddress);

Constructor Parameters:

1. **registryAddress:**
    - indicates the address of the registry contract which keeps the record of all the contract addresses in the ecosystem and the required configurations.

Contract Methods

execute()

Method name:

- execute

Method detail:

- This method executes the pullPayment of given subscription id of billing model for given pullPayment contract type.
- Internally this method first gets the address of the pullPayment contract from the pullPayment registry contract and calls the executePullPayment() method of that pullPayment contract.
- The executePullPayment() method of pullPayment contract further calls the execute method of Executor contract by passing the sender, receiver, settlement token address, payment token address and payment amount.
- Anyone can call this method by providing the valid parameters.

Method signature:

- function execute(string calldata _bmType, uint256 _subscriptionId)
  external
  override
  returns (uint256 pullPaymentID)

Method parameters:

1. **_bmType:**
   - Indicates the name of the pullPayment contract whose pullPayment is to execute. bmType is specified in the bytes32 format.
2. **_subscriptionId:**
   - Indicates the subscription id whose pullPayment is to execute.

Returned data:

- This method returns the unique pullPayment id which is just executed.

execute()

Method name:

- execute

Method detail:

- The execute method executes the pullPayment for the pullPayment contracts.
- Only the registered and granted pullPayment contracts can call this method.
- Internally this method first gets the amount of tokens from the customer, converts the payment tokens to settlement tokens, transfers the execution fee to the execution fee receiver which is in our case a token converter contract and then transfers the converted tokens to the merchant.
- It uses the quichwap router for swapping the tokens.
- This method always gets the execution fee in PMA tokens and transfers it to the Token Converter contract which contains the logic for converting PMA tokens to the LINK tokens. The Link tokens are used to top up the upkeeps when their balance falls below the minimum threshold.

Method signature:

- function execute(
  address settlementToken,
  address paymentToken,
  address from,
  address to,
  uint256 amount
)
  external
  virtual
  override
  returns (
   uint256 executionFee,
   uint256 userAmount,
   uint256 receiverAmount
)

Method parameters:

- **settlementToken:**
  - Indicates the token address in which the merchant wants to get paid in.
- **paymentToken:**
  - Indicates the payment token address in which the customer wants to pay for the subscription.
- **from:**
  - Indicates the customer address from which tokens are transferred to the merchant.
- **to:**

    - Indicates the merchant address.
- **amount:**
    - Indicates the subscription amount in settlement tokens. User pays the equal amount in the payment token.

Returned data:

- This method returns the following information-
    - **executionFee:**
        - Indicates the amount of PMA tokens charged for executing the pullpayment.
        - This fee is calculated based on the execution fee percentage.
        - ExecutionFee = (paymentAmount * executionFeePercent) / 10000
            - Hare, executionFeePercent = 500 (5%)
    - **userAmount:**
        - Indicates the amount of payment tokens that the user needs to pay for the pullpayment.
    - **receiverAmount:**
        - Indicates the amount of settlement tokens that the merchant will receive for the pullpayment.

getReceivingAmount()

Method name:

- getReceivingAmount

Method detail:

- This method gives the information about the required amount of payment tokens that subscribers will be charged for the pullpayment, the amount of execution fee and the amount of settlement tokens that the receiver will receive for given information.

Method signature:

- function getReceivingAmount(
  address _paymentToken,
  address _settlementToken,
  uint256 _amount
)
  public view virtual
  returns (
   uint256 receivingAmount,
   uint256 userPayableAmount,
   uint256 executionFee
)

Method parameters:

- **paymentToken:**
  - Indicates the payment token address in which the customer wants to pay for the subscription.
- **settlementToken:**
  - Indicates the token address in which the merchant wants to get paid in.
- **amount:**
  - Indicates the subscription amount in settlement tokens. User pays the equal amount in the payment token.

Returned data:

- This method returns the following information-
  - **executionFee:**
    - Indicates the amount of PMA tokens charged for executing the pullpayment.
    - This fee is calculated based on the execution fee percentage.
    - ExecutionFee = (paymentAmount * executionFeePercent) / 10000
      - Hare, executionFeePercent = 500 (5%)
  - **userAmount:**
    - Indicates the amount of payment tokens that the user needs to pay for the pullpayment.
  - **receiverAmount:**
    - Indicates the amount of settlement tokens that the merchant will receive for the pullpayment.

canSwapFromV2()

Method name:

- canSwapFromV2

Method detail:

- This method checks whether a from token can be converted to toToken or not.
- Returns true and swap path if there is path otherwise returns false.
- It returns two paths if there is no direct swap path through the PMA.
- This method ensures that we always go through the PMA token for the swap event if from token and to tokens are the same non-PMA tokens.

Method signature:

- function canSwapFromV2(address _fromToken, address _toToken)
  public
  view
  virtual
  returns (
  bool canSWap,
  bool isTwoPaths,
  address[] memory path1,
  address[] memory path2
)

Method parameters:

- _fromToken**:**
  - Indicates the payment token address in which the customer wants to pay for the subscription.
- _toToken**:**
  - Indicates the token address in which the merchant wants to get paid in.

Returned data:

- This method returns the following information-
  - canSWap**:**
    - This indicates whether there exists a swap path or not.
  - isTwoPaths:
    - This indicates whether there are two paths for the swap or not in order to convert the from token to toToken.
    - If the value of this is true, then we get the two paths i.e path1 and path2.
  - path1:
    - Indicates the only swap path for the tokens if there is only one swap path.
    - Indicates the first swap path for the tokens which is used to convert the from token to PMA tokens.
  - path2:
    - Indicates the second swap path if there are two swap paths.
    - It is empty when there is only one path.

- When there is a second path, the first token in this path is the PMA token which will be converted to the settlement token.

Flow of Execution:

1. First we set the main registry, pullPayment registry, pma token, dex router and factory addresses at the time of contract initialization.
2. The pullPayment contract who wants to execute their pullPayments for the subscription has to be registered on the pullPayment registry contract.
3. Anyone can execute the pullPayment for a given subscription using the external execute method, which requires the pullPayment contract name and the subscription id.
4. The external execute method gets the address of the pullPayment contract from the pullPayment Registry and creates the interface for the pullPayment contract.
5. Further, it calls the executePullPayment() method of the pullPayment contract.
6. The pullPayment contract further calls the execute method of the Executor contract to pull the payment from the customer and transfer it to the merchant.

Flow of Execution with pull payment contracts:

1. Pull Payment contract calls the execute() method of the Executor with the settlement token, payment token, subscriber address, merchant address and the subscription amount in settlement token.
2. The Executor gets the swap path for the conversion of the tokens if both payment and settlement tokens are not PMA tokens.
3. It then gets the equal worth of payment tokens from the subscriber.
4. Then swaps the payment tokens to the PMA tokens using the path1 in order to get the execution fee in PMA tokens.
5. Calculates the execution fee in PMA tokens and transfers it to the execution fee receiver which is Token Converter contract.
6. The Remaining PMA tokens are then converted to the settlement tokens and are transferred to the receiver.
7. Then we check if the upkeep contracts(Recurring contracts) have enough LINK tokens for the auto pullpayment execution.
8. If there are not enough LINK tokens, The Token Converter swaps the PMA tokens to LINK tokens and adds the fund for the calling upkeep.