SwapContract Documentation

##Functions

swapETHToTokenV3(tokenOut, amountOutMin, fee, sqrtPriceLimitX96)

{value: amount of ether }

@PARAMS: tokenOut: A string of characters represents the address of token which user wants to receive at the end of swap.

amountOutMin: Minimum amount tokenOut that user wants to receive after swap.

fee : A number which represents Uniswap fee per swap (always would be set to 3000)

sqrtPriceLimitX96: if the price of tokenOut during swap reaches to this amount, the swap will be reverted.

value : this is the built-in parameter that sends native coin (ETH) to the contract

@NOTICE: You can get amountOutMin by calling 'callEstimateAmountOut'

In order to get price of tokenOut but the amount that you get from this function might be higher than what we expect and you'll face 'Too Little received' error. My suggestion is to use lower price than what you get. For instance, if function returns 200, you use 195. Also, you can set it to 0 (zero) if you don't want user exactly specify the amount that he can receive.

@NOTICE: You can also set ‘sqrtPriceLimitX96’ **to 0 (zero) in order to ignore it. But if you want to use it, you must do several calculations.**

---

swapTokenToETHV3(tokenIn**,** amountIn**,** amountOutMin**,** fee**,** sqrtPriceLimitX96)

@PARAMS: **tokenIn: A string of characters represents the address of token which user wants to send to contract and receive ETH at the end.**

amountIn: A number which represents the amount of tokens you want to swap.

amountOutMin: A number which represents minimum ETH amount you want to receive.

fee : A number which represents Uniswap fee per swap (always would be set to 3000)

sqrtPriceLimitX96: if the price of ETH during swap reaches to this amount, the swap will be reverted.

@NOTICE: Before calling this function, you have to approve contract for ‘tokenIn' because contract then would be able to receive ‘amountIn’

@NOTICE: Same suggestions for ‘amountOutMin’ and ‘sqrtPriceLimitX96’.

---

swapTokenToTokenV3(tokenIn, tokenOut, amountIn, amountOutMin, fee, sqrtPriceLimitX96)

@PARAMS: **tokenIn: A string of characters represents the address of token which user wants to send to contract in order to swap.**

tokenOut: **A string of characters represents the address of token which user wants to receive at the end of swap.**

amountIn: A number which represents the amount of tokens of 'tokenIn' address you want to swap.

fee: **A number which represents Uniswap fee per swap (always would be set to 3000)**

sqrtPriceLimitX96: if the price of tokenOut during swap reaches to this amount, the swap will be reverted.

@NOTICE: Before calling this function, you have to approve contract for 'tokenIn' because contract would be able to receive 'amountIn'

---

callEstimateAmountOut(tokenIn, tokenOut, amountIn, fee, secondsAgo)

@PARAMS: **tokenIn**: A string of characters which represents the address of token you want to send for swap.

tokenOut: A string of characters which represents the address of token you want to receive after swap.

amountIn: A number which represents the number of tokens you want to swap

fee: A number which represents the Uniswap fee for swap. (always set it to 3000)

secondsAgo: A number which represents how many seconds in the past you want to look for price information. (you can set it to 10, means 10 seconds ago)

@NOTICE: When you use this function, you get the price of tokenOut in terms of tokenIn.

@NOTICE: You pass the result of this function as 'amountOutMin' for swap function. But consider this, the swap might not go well because the amount that user receives might be less than the price you showed as minimum. My suggestion is to always use lower price of what you get. If the price of ETH in terms of USDT is 2200$, instead, you should use 2100$.

---

addRecipient(recipient, amountToReceipt)

@PARAMS: recipient: A string of characters which represents the address of new receiver.

amountToReceipt: A number which will be the percentage of claim. i.e. 200 means 2%.

@NOTICE: this function is only owner, meaning only the contract owner can call this function.

withdrawTokenFromContract(token)

@PARAMS: token: A string of characters represents the address of token that you want to withdraw.

@NOTICE: caller must be owner of contract and the tokens will transfer to the owner address.

---

changeRecipientFee(recipient, newFeeAmount)

@PARAMS: recipient: The address that you want to change his fee percentage.

newFeeAmount: A number which will be the percentage of claim. i.e. 200 means 2%.

@NOTICE: First, a recipient should be added in order to change his fee percentage.

---

setFeeForTokens(numerator, denominator)

@DESCRIBE: this function would set a constant fee for unstable coins. i.e. $\frac{2}{100}$ **=> 2 would be numerator, 100 would be denominator**

@NOTICE: before lunching the project, you must call this function in order to have fee calculation for swaps. It is also only owner function

---

addStableCoinAddress(stableCoinAddress)

@PARAMS: stableCoinAddress: A string of characters which represents the address of a stable coin. i.e. USDC, DAI.

@NOTICE: before lunching the project, you should call this function in order to specify the stable coins that you want to put fees on them before swap.

---

changeOwner(newOwner)

@PARAMS: newOwner: A string of characters which represents the new owner address

changeFeeInUsd(newFee)

@DESCRIBE: This function changes the fee of stable coins in contract. For instance, 500000 means 0.5$, 1000000 means 1$

@NOTICE: This function is also only owner.

---

getRecipientById(id)

@PARAMS: id: A number which represents the id of recipient that is added.

@RETURN: it returns an array of these values =>

[recipientAddress , amountToReceipt , recipientId]

---

getRecipientByAddress(recipient)

@PARAMS: recipient: An address which represents the recipient address

@RETURN: it returns an array of these values =>

[recipientAddress , amountToReceipt , recipientId]

---

getAllRecipients()

@DESCRIBE: This function will return all of the recipients.

---

getAllStableCoins()

@DESCRIBE: This function will return all of the stable coins that are added to the contract.