Cyberscope

Documentation
Paxe Staking

July 2024

Created by © cyberscope

Table of Contents

Table of Contents **1**
LiquidFarming.sol **2**
constructor 2
deposit 2
claim 3
setReferrals 3
setTokenReceiver 3
setSpecialReferrerStakeReward 4
setSpecialReferrer 4
getPendingRewards 4
getReceiver 4
PaxeRestaking.sol **5**
constructor 5
restake 5
claim 5
pendingRewards 5
PaxeStandardStaking.sol **6**
constructor 6
poolLength 6
addPool 6
set 7
deposit 7
claim 7
withdraw 8
emergencyWithdraw 8
setTokenReceiver 8
getReceiver 8
Oracle.sol **9**
constructor 9
update 9
consult 9
OracleV3.sol **10**
constructor 10
consult 10
About Cyberscope **11**

LiquidFarming.sol

constructor

Constructor for initializing the staking contract.

```
constructor(
    PPAXE _pPAXE,
    address _PAXE,
    address _SAKAI,
    address _sakaiVaultV1,
    address _sakaiVaultV2,
    address _paxeTreasury,
    address _restakePool,
    address _paxeOracle,
    address _sakaiOracle,
    address _owner
) Ownable(_owner)
```

deposit

This function allows a user to deposit an equal value of SAKAI and PAXE tokens, with an optional referrer to participate in the referral program. It ensures that the staked amounts are sufficient and processes the staking by transferring the tokens and minting pPAXE tokens.

```
deposit(uint256 sakaiAmount, uint256 paxeAmount) external payable
deposit(uint256 sakaiAmount, uint256 paxeAmount, address referrer)
external payable
```

1. sakaiAmount (uint256): The amount of SAKAI tokens to stake. This must be a positive value and not zero.
2. paxeAmount (uint256): The amount of PAXE tokens to stake. This must be a positive value and not zero.
3. referrer (address): The address of the referrer. This cannot be the same as the sender's address (msg.sender), and the address must be valid (non-zero).

claim

The claim function in the PaxeStaking contract is designed to calculate and distribute the pending rewards for a user, delete completed stakes, and distribute the associated fees to the treasury and restake pool. This function ensures that the rewards are properly distributed and any fees are appropriately handled.

```
claim() external updateOracle
```

setReferrals

The setReferrals function is callable only by the owner and updates the referrals in a referral program. This function's purpose is to initialize the referral program to the off-chain state.

```
struct ReferralsInit {
    address referrer;
    address[] referees;
}

setReferrals(ReferralProgram.ReferralsInit[] calldata refers) external
onlyOwner
```

1. refers (ReferralProgram.ReferralsInit[]): An array with referees of a referrer.

setTokenReceiver

The setTokenReceiver function is callable only by the owner and sets the receiver address for a specific user.

```
setTokenReceiver(address user, address receiver) external onlyOwner
```

1. user (address): Parameter specifying the user for whom to set the receiver address.
2. receiver (address): Parameter specifying the receiver address of the user’s rewards.

setSpecialReferrerStakeReward

The setSpecialReferrerStakeReward function sets the referral's stake rewards for the special referrers. May be set between 0.01% up to 5%. The function is callable only by the owner.

```
setSpecialReferrerStakeReward(uint256 stakeReward) external onlyOwner
```

1. stakeReward (uint256): The stake rewards that the special referrer will get when a referee deposits.

setSpecialReferrer

The setSpecialReferrer function allows the owner of the contract to designate an address as a special referrer or remove that designation. It emits an event whenever the status of an address is changed, and it reverts if the address already has the specified status.

```
setSpecialReferrer(address referrer, bool value) external onlyOwner
```

1. referrer (address): The address which will be set/unset as a special referrer address.
2. value (bool): Wether will be set or unset.

setSendPaxeTo

The setSendPaxeTo function allows the owner of the contract to set an address to send the deposited PAXE to, otherwise if this address is equal to the zero address, the PAXE will be sent to the dead address.

```
setSendPaxeTo(address destination) external onlyOwner
```

1. destination (address): The address which will be receiving the deposited PAXE.

getPendingRewards

The getPendingRewards function calculates the pending rewards for a given user. The function takes into account whether the user is a special referrer, calculates rewards from referral stakes, and includes rewards from the user's own stakes.

```
getPendingRewards(address user) public view returns (uint256 rewards)
```

1. user (address): Parameter specifying the user for whom pending rewards are being calculated.

getReceiver

The getReceiver function returns the address that the contract will send the rewards to instead of the user.

```
getReceiver(address user) public view returns (address)
```

1. user (address): Parameter specifying the user for whom to get the receiver address.

PaxeRestaking.sol

constructor

Constructor for initializing the restaking contract.

```
constructor(address _pPAXE, address _PAXE)
```

restake

The restake function allows users to stake pPAXE tokens. It records the staking information by creating a new StakeInfo struct and storing it in the stakeInfo mapping for the user. It also emits a Restake event to notify listeners about the staking action.

```
restake(uint256 amount) external
```

1. amount (uint256): The amount of pPAXE tokens that the user wants to stake.

claim

The claim function calculates pending rewards for all active stakes of the caller, updates the stake information accordingly, and transfers the calculated rewards to the caller. It also emits a Claim event to log the reward distribution.

```
claim() external
```

pendingRewards

The pendingRewards function calculates the total pending rewards for a specific user across all their active stakes.

```
pendingRewards(address user) public view returns (uint256 rewards)
```

1. user (address): Parameter specifying the user for whom pending rewards are being calculated.

PaxeStandardStaking.sol

constructor

The constructor of the PaxeStandardStaking contract initializes several important parameters and performs validations to ensure that the contract is properly configured upon deployment.

```
constructor(address _pPAXE, address _PAXE, address _paxeOracle, address
initialOwner) Ownable(initialOwner)
```

poolLength

The poolLength function is a simple getter function that retrieves the number of staking pools currently defined in the contract.

the time of the claim.

```
poolLength() public view returns (uint256 pools)
```

addPool

The addPool function in the PaxeStandardStaking contract allows the owner to add a new staking pool with specified parameters.

```
addPool(uint256 _apr, IERC20 _token, address oracle) external onlyOwner
```

1. _apr (uint256): An uint256 representing the Annual Percentage Rate (APR) of the pool, which will be converted to an internal precision (APR_PRECISION_DECIMALS).
2. _token (IERC20): An instance of the IERC20 interface representing the token used for staking in the new pool.
3. oracle (address): An address representing the oracle contract used to fetch token prices and validate the token pair.

set

The set function allows the contract owner to update the Annual Percentage Rate (APR) for a specific pool.

```
set(uint256 _pid, uint256 _apr) external onlyOwner
```

1. _pid (uint256): An uint256 representing the Pool ID (index) of the pool in the poolInfo array that needs its APR updated.
2. _apr (uint256): An uint256 representing the new Annual Percentage Rate (APR) that will be applied to the pool.

deposit

The deposit function contract allows users to deposit tokens into a specified staking pool.

```
deposit(uint256 _pid, uint256 _amount) external nonReentrant
updateOracle(_pid)
```

1. pid (uint256): Represents the Pool ID (index) in the poolInfo array to which tokens are being deposited.
2. amount (uint256): Specifies the amount of tokens to deposit to the pool.

claim

The claim function is designed to calculate and send pending rewards to a user for a specific staking pool.

```
claim(uint256 _pid) external
```

1. pid (uint256): Represents the Pool ID (index) in the poolInfo array from which tokens are being claimed.

withdraw

The withdraw function allows a user to withdraw tokens from a specified pool.

```
withdraw(uint256 _pid, uint256 _amount) external nonReentrant
```

1. pid (uint256): Represents the Pool ID (index) in the poolInfo array from which tokens are being withdrawn.
2. amount (uint256): Specifies the amount of tokens to withdraw from the pool.

emergencyWithdraw

The emergencyWithdraw function allows users to immediately withdraw tokens from a specified pool without claiming any pending rewards.

```
emergencyWithdraw(uint256 _pid, uint256 _amount) external nonReentrant
```

1. pid (uint256): Represents the Pool ID (index) in the poolInfo array from which tokens are being withdrawn.
2. amount (uint256): Specifies the amount of tokens to withdraw from the pool.

setTokenReceiver

The setTokenReceiver function is callable only by the owner and sets the receiver address for a specific user.

```
setTokenReceiver(address user, address receiver) external onlyOwner
```

3. user (address): Parameter specifying the user for whom to set the receiver address.
4. receiver (address): Parameter specifying the receiver address of the user's rewards.

getReceiver

The getReceiver function returns the address that the contract will send the rewards to instead of the user.

```
getReceiver(address user) public view returns (address)
```

1. user (address): Parameter specifying the user for whom to get the receiver address.

Oracle.sol

constructor

The constructor function initializes an instance of the Oracle contract by fetching and setting up information from a Uniswap V2 pair.

```
constructor(address factory, address tokenA, address tokenB)
```

1. factory (address): The address of the Uniswap V2 factory contract used to retrieve the pair address.
2. tokenA (address): Address of the first token in the Uniswap V2 pair.
3. tokenB (address): Address of the second token in the Uniswap V2 pair.

update

The update function in the Oracle contract updates the average prices based on the current cumulative prices fetched from the Uniswap V2 pair.

```
update() external
```

consult

The consult function retrieves the estimated output amount for a given input amount of a specified token.

```
consult(address token, uint256 amountIn) external view returns (uint256 amountOut)
```

1. token (address): Address of the token for which the output amount is being estimated.
2. amountIn (uint256): Input amount of tokens for which the output amount is to be estimated.

OracleV3.sol

constructor

Initializes the contract with the address of a specific Uniswap V3 pool.

```
constructor(address _pool)
```

1. _pool (address): Address of the Uniswap V3 pool for which the oracle is being created.

consult

Estimates the output amount for a given input amount of a specified token using data from the Uniswap V3 pool.

```
consult(address token, uint256 amountIn) external view override returns (uint256 amountOut)
```

1. token (address): Address of the token for which the output amount is being estimated.
2. amountIn (uint256): Input amount of tokens for which the output amount is to be estimated.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io