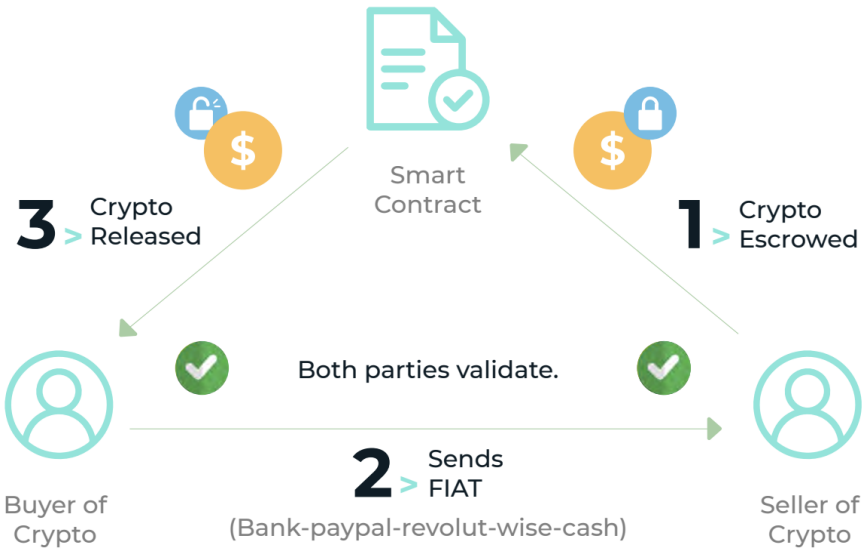
paydece

# Paydece Escrow

## Project Overview

paydece is the needed infrastructure so that users can exchange crypto to fiat P2P using a smart contract escrow trading privately without needing a central institution validation. With Ethereum technology paydece allows users to trade with the vision of Bitcoin. The decentralized Dapp allows users to buy or sell crypto using FIAT currencies operating from their self-custodial wallet.

Users connect to the Dapp with their wallet where they own their private keys (they have total control over their funds) and will block their funds in a smart contract that will hold the crypto until the FIAT payment is done. Crypto funds are only released from the smart contract escrow when both sides validate the transaction was successfully completed.



# Characteristics of the product

1) **Use app.paydece.io and connect your wallet.**

Connect to the app using your self-custodial EVM-compatible wallet. Enter the password and sign in to the app.

2) **Buy or sell by taking an order from the marketplace.**

For trading, you must ensure that you have your own telegram account so that you can contact the counterparty during the transaction.

3) **The crypto is escrowed in the smart contract**

The crypto seller must approve and create the escrow using the smart contract. Crypto will be escrowed and will not move from the contract until both sides validate.

4) **The buyer must send the FIAT to the counterparty.**

Once the funds are escrowed the crypto buyer sends the FIAT money to the crypto seller, using the previously agreed payment method. When he sends the payment, he must click as paid in the app.

5) **The crypto seller confirms the FIAT payment.**

Once the crypto seller receives the FIAT payment he will click as paid in the app, and the funds will be automatically released from the smart contract to the crypto buyer wallet.

# Disputes

If there is any problem during the transaction, meaning that the funds are escrowed, but one of the sides doesn't validate, a dispute is raised. paydece at the first stage (resolution of conflicts) can solve it and decide the side the funds will move to. If the dispute escalates and a complete legal framework is needed Kleros.io, a decentralized arbitration system will be used for finding the final resolution.

3

# Technical Information

This smart contract implements an escrow service for cryptocurrency transactions. It allows the creation of an escrow for a given order, where a buyer and a seller can deposit their funds, which are held by the contract until the transaction is completed or a dispute is resolved. The contract can be used to hold funds in different currencies, as long as they are ERC20 tokens whitelisted.

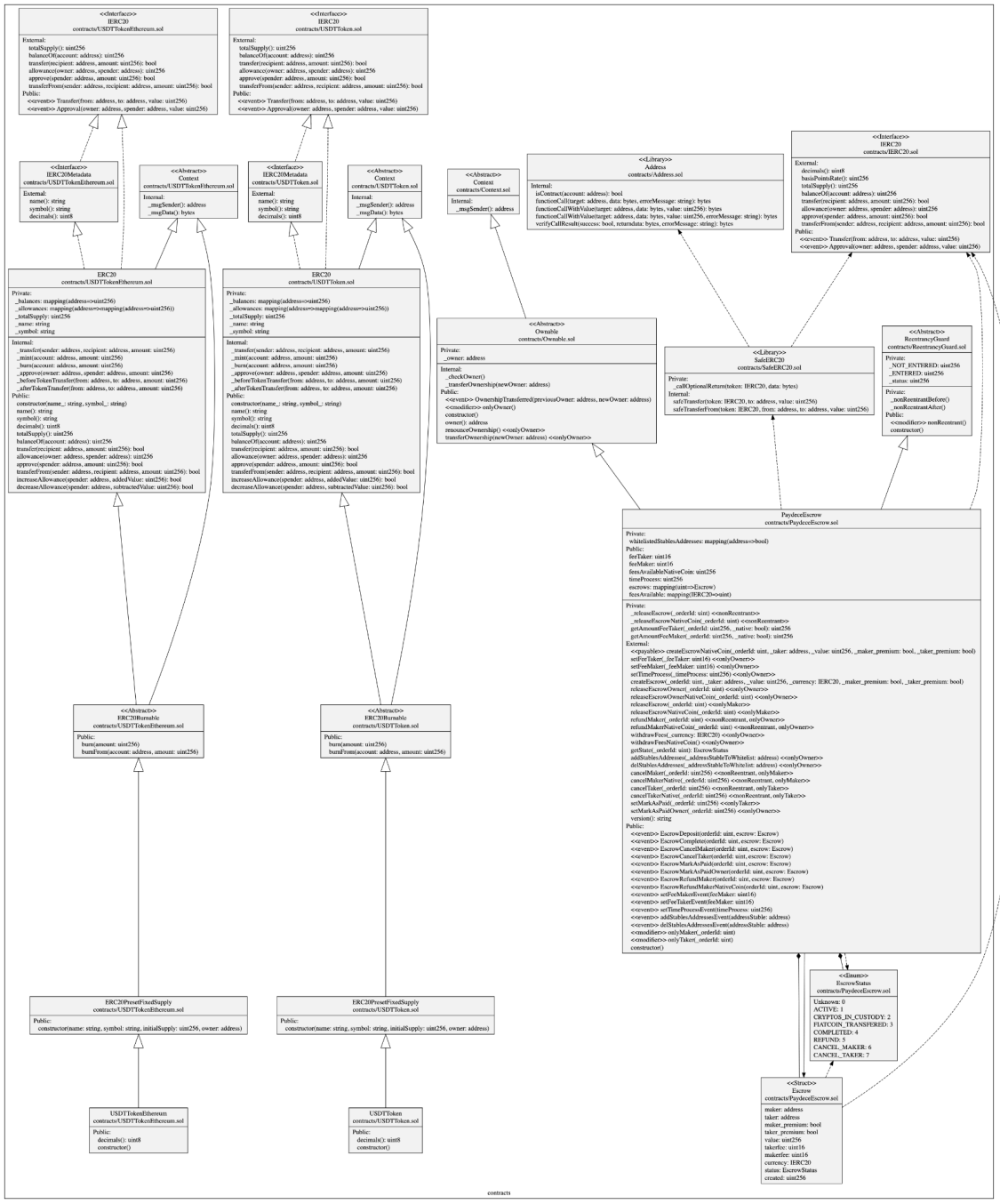
The contract defines a structure called Escrow to represent an escrow, which contains the addresses of the buyer and seller, the value of the transaction, the commissions for the buyer and seller, the currency used, the premium information and the status of the escrow. The contract also maintains a correspondence between the order ID and the escrow to store the escrows that have been created.

The contract has a whitelistedStablesAddresses assignment to store a list of whitelisted stablecoin addresses and checks that the currency used in an escrow is a whitelisted stablecoin before creating the escrow.

There is also a feesAvailable assignment to track fees that can be withdrawn, and a feesAvailableNativeCoin variable to track fees that can be withdrawn in the native currency (presumably ETH).

The contract has a modifier onlyBuyer and onlySeller to restrict certain functions to be called only by the escrow buyer or seller, respectively. It also has an onlyOwner modifier to restrict certain functions to be called only by the contract owner. The contract has a base contract Ownable, which provides an onlyOwner modifier and an owner address to store the address of the contract owner.

# Diagram


<<Interface>>
IERC20
contracts/USDTTokenEthereum.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(recipient: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(sender: address, recipient: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Interface>>
IERC20
contracts/USDTToken.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(recipient: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(sender: address, recipient: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Interface>>
IERC20Metadata
contracts/USDTTokenEthereum.sol
External:
name(): string
symbol(): string
decimals(): uint8
<<Abstract>>
Context
contracts/USDTTokenEthereum.sol
Internal:
_msgSender(): address
_msgData(): bytes
<<Interface>>
IERC20Metadata
contracts/USDTToken.sol
External:
name(): string
symbol(): string
decimals(): uint8
<<Abstract>>
Context
contracts/USDTToken.sol
Internal:
_msgSender(): address
_msgData(): bytes
<<Abstract>>
Context
contracts/Context.sol
Internal:
_msgSender(): address
<<Library>>
Address
contracts/Address.sol
Internal:
isContract(account: address): bool
functionCall(target: address, data: bytes, errorMessage: string): bytes
functionCallWithValue(target: address, data: bytes, value: uint256): bytes
functionCallWithValue(target: address, data: bytes, value: uint256, errorMessage: string): bytes
verifyCallResult(success: bool, returndata: bytes, errorMessage: string): bytes
<<Interface>>
IERC20
contracts/IERC20.sol
External:
decimals(): uint8
basisPointsRate(): uint256
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(recipient: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(sender: address, recipient: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
ERC20
contracts/USDTTokenEthereum.sol
Private:
_balances: mapping(address=>uint256)
_allowances: mapping(address=>mapping(address=>uint256))
_totalSupply: uint256
_name: string
_symbol: string
Internal:
_transfer(sender: address, recipient: address, amount: uint256)
_mint(account: address, amount: uint256)
_burn(account: address, amount: uint256)
_approve(owner: address, spender: address, amount: uint256)
_beforeTokenTransfer(from: address, to: address, amount: uint256)
_afterTokenTransfer(from: address, to: address, amount: uint256)
Public:
constructor(name_: string, symbol_: string)
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(recipient: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(sender: address, recipient: address, amount: uint256): bool
increaseAllowance(spender: address, addedValue: uint256): bool
decreaseAllowance(spender: address, subtractedValue: uint256): bool
ERC20
contracts/USDTToken.sol
Private:
_balances: mapping(address=>uint256)
_allowances: mapping(address=>mapping(address=>uint256))
_totalSupply: uint256
_name: string
_symbol: string
Internal:
_transfer(sender: address, recipient: address, amount: uint256)
_mint(account: address, amount: uint256)
_burn(account: address, amount: uint256)
_approve(owner: address, spender: address, amount: uint256)
_beforeTokenTransfer(from: address, to: address, amount: uint256)
_afterTokenTransfer(from: address, to: address, amount: uint256)
Public:
constructor(name_: string, symbol_: string)
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(recipient: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(sender: address, recipient: address, amount: uint256): bool
increaseAllowance(spender: address, addedValue: uint256): bool
decreaseAllowance(spender: address, subtractedValue: uint256): bool
<<Abstract>>
Ownable
contracts/Ownable.sol
Private:
_owner: address
Internal:
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor()
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
<<Library>>
SafeERC20
contracts/SafeERC20.sol
Private:
_callOptionalReturn(token: IERC20, data: bytes)
Internal:
safeTransfer(token: IERC20, to: address, value: uint256)
safeTransferFrom(token: IERC20, from: address, to: address, value: uint256)
<<Abstract>>
ReentrancyGuard
contracts/ReentrancyGuard.sol
Private:
_NOT_ENTERED: uint256
_ENTERED: uint256
_status: uint256
Private:
_nonReentrantBefore()
_nonReentrantAfter()
Public:
<<modifier>> nonReentrant()
constructor()
PaydeceEscrow
contracts/PaydeceEscrow.sol
Private:
whitelistedStablesAddresses: mapping(address=>bool)
Public:
feeTaker: uint16
feeMaker: uint16
feesAvailableNativeCoin: uint256
timeProcess: uint256
escrows: mapping(uint=>Escrow)
feesAvailable: mapping(IERC20=>uint)
Private:
_releaseEscrow(_orderId: uint) <<nonReentrant>>
_releaseEscrowNativeCoin(_orderId: uint) <<nonReentrant>>
getAmountFeeTaker(_orderId: uint256, _native: bool): uint256
getAmountFeeMaker(_orderId: uint256, _native: bool): uint256
External:
<<payable>> createEscrowNativeCoin(_orderId: uint, _taker: address, _value: uint256, _maker_premium: bool, _taker_premium: bool)
setFeeTaker(_feeTaker: uint16) <<onlyOwner>>
setFeeMaker(_feeMaker: uint16) <<onlyOwner>>
setTimeProcess(_timeProcess: uint256) <<onlyOwner>>
createEscrow(_orderId: uint, _taker: address, _value: uint256, _currency: IERC20, _maker_premium: bool, _taker_premium: bool)
releaseEscrowOwner(_orderId: uint) <<onlyOwner>>
releaseEscrowOwnerNativeCoin(_orderId: uint) <<onlyOwner>>
releaseEscrow(_orderId: uint) <<onlyMaker>>
releaseEscrowNativeCoin(_orderId: uint) <<onlyMaker>>
refundMaker(_orderId: uint) <<nonReentrant, onlyOwner>>
refundMakerNativeCoin(_orderId: uint) <<nonReentrant, onlyOwner>>
withdrawFees(_currency: IERC20) <<onlyOwner>>
withdrawFeesNativeCoin() <<onlyOwner>>
getState(_orderId: uint): EscrowStatus
addStablesAddresses(_addressStableToWhiteList: address) <<onlyOwner>>
delStablesAddresses(_addressStableToWhiteList: address) <<onlyOwner>>
cancelMaker(_orderId: uint256) <<nonReentrant, onlyMaker>>
cancelMakerNative(_orderId: uint256) <<nonReentrant, onlyMaker>>
cancelTaker(_orderId: uint256) <<nonReentrant, onlyTaker>>
cancelTakerNative(_orderId: uint256) <<nonReentrant, onlyTaker>>
setMarkAsPaid(_orderId: uint256) <<onlyTaker>>
setMarkAsPaidOwner(_orderId: uint256) <<onlyOwner>>
version(): string
Public:
<<event>> EscrowDeposit(orderId: uint, escrow: Escrow)
<<event>> EscrowComplete(orderId: uint, escrow: Escrow)
<<event>> EscrowCancelMaker(orderId: uint, escrow: Escrow)
<<event>> EscrowCancelTaker(orderId: uint, escrow: Escrow)
<<event>> EscrowMarkAsPaid(orderId: uint, escrow: Escrow)
<<event>> EscrowMarkAsPaidOwner(orderId: uint, escrow: Escrow)
<<event>> EscrowRefundMaker(orderId: uint, escrow: Escrow)
<<event>> EscrowRefundMakerNativeCoin(orderId: uint, escrow: Escrow)
<<event>> setFeeMakerEvent(feeMaker: uint16)
<<event>> setFeeTakerEvent(feeMaker: uint16)
<<event>> setTimeProcessEvent(timeProcess: uint256)
<<event>> addStablesAddressesEvent(addressStable: address)
<<event>> delStablesAddressesEvent(addressStable: address)
<<modifier>> onlyMaker(_orderId: uint)
<<modifier>> onlyTaker(_orderId: uint)
constructor()
<<Abstract>>
ERC20Burnable
contracts/USDTTokenEthereum.sol
Public:
burn(amount: uint256)
burnFrom(account: address, amount: uint256)
<<Abstract>>
ERC20Burnable
contracts/USDTToken.sol
Public:
burn(amount: uint256)
burnFrom(account: address, amount: uint256)
<<Enum>>
EscrowStatus
contracts/PaydeceEscrow.sol
Unknown: 0
ACTIVE: 1
CRYPTOS_IN_CUSTODY: 2
FIATCOIN_TRANSFERED: 3
COMPLETED: 4
REFUND: 5
CANCEL_MAKER: 6
CANCEL_TAKER: 7
<<Struct>>
Escrow
contracts/PaydeceEscrow.sol
maker: address
taker: address
maker_premium: bool
taker_premium: bool
value: uint256
takerfee: uint16
makerfee: uint16
currency: IERC20
status: EscrowStatus
created: uint256
ERC20PresetFixedSupply
contracts/USDTTokenEthereum.sol
Public:
constructor(name: string, symbol: string, initialSupply: uint256, owner: address)
ERC20PresetFixedSupply
contracts/USDTToken.sol
Public:
constructor(name: string, symbol: string, initialSupply: uint256, owner: address)
USDTTokenEthereum
contracts/USDTTokenEthereum.sol
Public:
decimals(): uint8
constructor()
USDTToken
contracts/USDTToken.sol
Public:
decimals(): uint8
constructor()
contracts

5

# Functions & methods

### cancelMaker ()

Allows the cryptocurrency seller (maker) to cancel the fund custody once the time specified in the #setTimeProcess method has passed and the buyer (taker) has NOT marked the fiat payment as paid or confirmed the fiat transfer (#SetMarkAsPaid). Executing this method will result in a refund of the cryptocurrencies to the seller. The available time for the buyer (taker) is currently set to 45 minutes from the moment the cryptocurrencies are placed in custody.

### cancelTaker ()

This method allows the cryptocurrency buyer (taker) to cancel the transaction and refund the funds to the seller (maker) if they decide not to proceed with the transaction. The buyer can execute this method once the cryptocurrencies are in custody.

### getstate ()

Method that allows to obtain the current status of the escrow by inserting the ID.

### CancelMakerNative ()

Same method as the cancelmaker, exclusively for canceling escrows in the native coin of the blockchain in question.

### CancelTakerNative ()

Same method as the canceltaker, exclusively for canceling escrows in the native coin of the blockchain in question.

### addStablesAddresses ()

Allows the contract owner to add new tokens that are accepted by the contract.

### createEscrow ()

A method that places cryptocurrencies in custody and is executed by the cryptocurrency seller. The parameters sent to this method include: idescrow, crypto-amount, stable address, and the buyer's (taker's) wallet. With the latest implementations, additional data is sent to inform the contract if either of the parties (maker or taker) are premium users or subscribers. If they are subscribers, the contract will not charge them the established fee.

### createEscrowNativeCoin ()

Same method as the previous one, exclusively for creating escrows in the native coin of the blockchain in question.

### delStablesAddresses ()

Allows the contract owner to remove tokens accepted by the contract.

### refundMaker ()

Method exclusively executable by the contract owner that allows refunding funds to the cryptocurrency seller (maker) when a dispute arises and paydece determines that the funds should be returned.

### refundMakerNativeCoin ()

Same method as the previous one, exclusively for refunding cryptocurrencies to the seller when applicable, in the native coin of the blockchain in question.

### releaseEscrow ()

Method executable by the cryptocurrency seller (maker) to proceed with releasing the cryptocurrencies in favor of the buyer (taker).

### releaseEscrowNativeCoin ()

Same method as the previous one, exclusively for releasing cryptocurrencies to the buyer when dealing with a native coin.

**releaseEscrowOwner ()**

Method exclusively executable by the contract owner to release funds to the cryptocurrency buyer (taker) when a dispute arises and paydece determines that the funds should be released.

**releaseEscrowOwnerNativeCoin ()**

Same method as the previous one, exclusively for releasing cryptocurrencies to the buyer by the contract owner when dealing with a native coin and it's necessary.

**renounceOwnership ()**

Method to relinquish ownership of the contract if necessary.

**setFeeMaker ()**

Method exclusively executable by the contract owner to configure the fee charged to the cryptocurrency seller (maker). This configuration is adjustable between the values of 0.1% to 1% maximum.

**setFeeTaker ()**

Method exclusively executable by the contract owner to configure the fee charged to the cryptocurrency buyer (taker). This configuration is adjustable between the values of 0.1% to 1% maximum.

**SetMarkAsPaid ()**

Method exclusively executable by the cryptocurrency buyer (taker) to confirm the fiat money transfer.

**SetMarkAsPaidOwner ()**

Method exclusively executable by the contract owner to confirm the fiat money transfer when issues arise and the buyer cannot perform it.

**setTimeProcess ()**

Method that allows the contract owner to configure the time after which the cryptocurrency seller (maker) could cancel the transaction and be refunded the funds, provided that the buyer hasn't marked it as paid within that timeframe (#setMarkAsPaid). The current time is set to 45 minutes.

**transferOwnership ()**

Method that allows transferring the ownership of the contract if necessary.

**withdrawFees ()**

Method that allows withdrawing fees from different ERC20 tokens.

**withdrawFeesNativeCoin ()**

Method that allows withdrawing fees generated in the native coin.

# Events

**EscrowDeposit ()**

The event is issued when the Escrow is created and the deposit is made in custody.

**EscrowComplete ()**

The event is issued when the Escrow is completed and the coins in custodial are released.

**EscrowRefundMaker ()**

The event is issued when the cryptos were refunded to the maker after a dispute.

**EscrowRefundMakerNativeCoin ()**

The event is the same as the previous but with a native coin.

**EscrowCancelMaker ()**

The event is issued when the escrow is canceled by the maker of an ERC-20 or a native escrow..

**EscrowCancelTaker ()**

The event is issued when the escrow is canceled by the taker in an ERC-20 or a native escrow..

**EscrowMarkAsPaid ()**

The event is issued when the taker mark is paid.

**EscrowMarkAsPaidOwner ()**

The event is issued when the contract owner marks as paid.

**setFeeMaker ()**

The event is issued when the contract owner updates the maker fee.

**setFeeTaker ()**

The event is issued when the contract owner updates the taker fee.

**setTimeProcess ()**

The event is issued when the contract owner updates the time process.

**addStablesAddresses ()**

The event is issued when the contract owner adds an ERC20 token to the whitelist.

### delStablesAddresses ()

The event is issued when the contract owner deletes an ERC20 token from the whitelist.