Solidity API

Factory

owner

```
address owner
```

The address of the owner of the contract

router

```
address router
```

The address of the {Router} contract

TokenAdded

```
event TokenAdded(address newToken, address pool, address poolRewards, address parifiRewards)
```

Indicates that a new supported ERC20 token for trading was added

SetRouterForPoolAndRewards

```
event SetRouterForPoolAndRewards(address pool, address poolRewards, address parifiRewards)
```

Indicates that a new router has been set for a pool contract and 2 rewards contracts

UpdateParams

```
event UpdateParams(uint256 minDepositTime, uint256 utilizationMultiplier, uint256 maxParifi, uint256 w
```

Indicates that new params have been set for a pool contract

constructor

```
constructor() public
```

setOwner

```
function setOwner(address _newOwner) external
```

Changes owner's address

Parameters

Name	Type	Description
_newOwner	address	New owner's address

setRouter

```
function setRouter(address _router) external
```

Changes router's address

Parameters

Name	Type	Description
_router	address	New router's address NOTE: Should be called at the very beginning

addToken

```
function addToken(address _currency, uint8 _decimals, uint256 _share) external
```

Add support for trading with new ERC20 token

Deploys new pool of the currency, new rewards contract for that currency pool and a new rewards contract for global parifi pool and gives router control over them

Parameters

Name	Type	Description
_currency	address	Address of added ERC20 token
_decimals	uint8	Decimals of added token
_share	uint256	Pool share of added token

setRouterForPoolAndRewards

```
function setRouterForPoolAndRewards(address _currency, address _router) external
```

Change router in contracts deployed through (and therefore owned by) the factory

Parameters

Name	Type	Description

Name	Type	Description
_currency	address	Token which router should be changed
_router	address	New router address

setParamsPool

```
function setParamsPool(address _currency, uint256 _minDepositTime, uint256 _utilizationMultiplier, uin
```

Change pool parameters in pool deployed through (and therefore owned by) the factory

Parameters

Name	Type	Description
_currency	address	Token which pool's parameters should be changed
_minDepositTime	uint256	Minimum deposit time
_utilizationMultiplier	uint256	Utilisation Multiplier
_maxParifi	uint256	Maximum amount of ether that can be stored in the pool
_withdrawFee	uint256	Withdraw fee

onlyOwner

```
modifier onlyOwner()
```

Allows only the owner of the contract to call functions

Oracle

Connects with the backend for position settlement

owner

```
address owner
```

The address of the owner of the contract

router

```
address router
```

The address of the {Router} contract

darkOracle

```
address darkOracle
```

The address of the backend

treasury

```
address treasury
```

The address of the {Treasury} contract

trading

```
address trading
```

The address of the {Trading} contract

requestsPerFunding

```
uint256 requestsPerFunding
```

The number of requests that can be processes before next payment to the oracle

If `requestsPerFunding` requests were processed the treasury transfers funds to the oracle the services are stopped (paused)

costPerRequest

```
uint256 costPerRequest
```

The default cost of a single request is 0.0006 ETH

requestsSinceFunding

```
uint256 requestsSinceFunding
```

Conter for requests processes sinse the funding

SettlementError

```
event SettlementError(address user, address currency, bytes32 productId, bool isLong, string reason)
```

Indicates that an error occured while settling the position request

constructor

```
constructor() public
```

setOwner

```
function setOwner(address newOwner) external
```

Sets the address of the owner of the contract

Parameters

Name	Type	Description
newOwner	address	The address of a the new owner

setRouter

```
function setRouter(address _router) external
```

Sets the address of the router to use

Gets addresses of {Trading}, {Treasury} and backend from the router and initializes values using them

Parameters

Name	Type	Description
_router	address	The address of the router to use

setParams

```
function setParams(uint256 _requestsPerFunding, uint256 _costPerRequest) external
```

Sets the number of requests waiting for funding and the cost of a sigle request

Parameters

Name	Type	Description
_requestsPerFunding	uint256	The number of requests waiting for funding
_costPerRequest	uint256	The cost of a single request

settleStopOrders

```
function settleStopOrders(address[] users, bytes32[] productIds, address[] currencies, bool[] directic
```

Settles stop-loss positions based on orders of multiple users

Parameters

Name	Type	Description
users	address[]	Batch of positions' owners
productIds	bytes32[]	Batch of positions' products
currencies	address[]	Batch of positions' tokens
directions	bool[]	Batch of positions' directions
stops	uint64[]	Batch of positions' stops

settleTakeOrders

```
function settleTakeOrders(address[] users, bytes32[] productIds, address[] currencies, bool[] directic
```

Settles take-profit positions based on orders of multiple users

Parameters

Name	Type	Description
users	address[]	Batch of positions' owners
productIds	bytes32[]	Batch of positions' products
currencies	address[]	Batch of positions' tokens
directions	bool[]	Batch of positions' directions
takes	uint64[]	Batch of positions' takes

settleOrders

```
function settleOrders(address[] users, bytes32[] productIds, address[] currencies, bool[] directions,
```

Settles standard positions based on orders of multiple users

Parameters

Name	Type	Description
users	address[]	Batch of positions' owners
productIds	bytes32[]	Batch of positions' products
currencies	address[]	Batch of positions' tokens
directions	bool[]	Batch of positions' directions
prices	uint256[]	Batch of positions' prices

settleLimits

```
function settleLimits(address[] users, bytes32[] productIds, address[] currencies, bool[] directions,
```

Closes positions and settles limits

Parameters

Name	Type	Description
users	address[]	Batch of positions' owners
productIds	bytes32[]	Batch of positions' products
currencies	address[]	Batch of positions' tokens
directions	bool[]	Batch of positions' directions
prices	uint256[]	Batch of closing prices

liquidatePositions

```
function liquidatePositions(address[] users, bytes32[] productIds, address[] currencies, bool[] direct
```

Liquidates positions of multiple users

Parameters

Name	Type	Description
users	address[]	Batch of positions' owners
productIds	bytes32[]	Batch of positions' products
currencies	address[]	Batch of positions' tokens

Name	Type	Description
directions	bool[]	Batch of positions' directions
prices	uint256[]	Batch of closing prices

_tallyOracleRequests

```
function _tallyOracleRequests(uint256 newRequests) internal
```

Sends funds to the backend if the number of processed requests is greater than the initial provided number of requests

onlyOwner

```
modifier onlyOwner()
```

Allows only the owner of the contract to call the function

onlyDarkOracle

```
modifier onlyDarkOracle()
```

Allows only the backend to call the function

Pool

owner

```
address owner
```

The address of the owner of this contract

router

```
address router
```

The address of the {Router} contract

trading

```
address trading
```

The address of the {Trading} contract

rewards

```
address rewards
```

The address of the {Rewards} contract

withdrawFee

```
uint256 withdrawFee
```

Withdrawing funds from the pool costs 0.3% extra fee

currency

```
address currency
```

The address of the currency (token) stored in the pool

utilizationMultiplier

```
uint256 utilizationMultiplier
```

Utilization multiplier (in Basis Points)

maxParifi

```
uint256 maxParifi
```

The maximum amount of ether that can be stored in the pool

balances

```
mapping(address => uint256) balances
```

*The mapping from account's address to the amount of LP tokens he got for deposit These are *not* amounts of tokens the user transferred into the pool!*

totalSupply

```
uint256 totalSupply
```

The total amount of *LP* tokens stored in the pool

lastDeposited

```
mapping(address => uint256) lastDeposited
```

The mapping from account's address to the time this account made his latest stake

minDepositTime

```
uint256 minDepositTime
```

The minimum time from deposit to withdrawal

openInterest

```
uint256 openInterest
```

The interest right after the position was opened

UNIT

```
uint256 UNIT
```

Decimals correction

Deposit

```
event Deposit(address user, address currency, uint256 amount, uint256 plpAmount)
```

Indicated that funds have been deposited into the pool

Withdraw

```
event Withdraw(address user, address currency, uint256 amount, uint256 plpAmount)
```

Indicates that funds have been withdrawn from the pool

constructor

```
constructor(address _currency) public
```

setOwner

```
function setOwner(address newOwner) external
```

Seths the address of the new owner of the contract

Parameters

Name	Type	Description
newOwner	address	The address of the new owner of the contact

setRouter

```
function setRouter(address _router) external
```

Sets the address of the new router used in the contract

Parameters

Name	Type	Description
_router	address	The address of the new router

setParams

```
function setParams(uint256 _minDepositTime, uint256 _utilizationMultiplier, uint256 _maxParifi, uint25
```

Changes crusial variables of the pool

Parameters

Name	Type	Description
_minDepositTime	uint256	A new minimal time from token deposit till token withdrawal
_utilizationMultiplier	uint256	A new utilization multiplier (in Basis Points)
_maxParifi	uint256	A new maximum amount of ether that can be stored in the pool
_withdrawFee	uint256	A new fee for tokens withdrawal

updateOpenInterest

```
function updateOpenInterest(uint256 amount, bool isDecrease) external
```

Updates open interest. Increase or decrease it.

Parameters

Name	Type	Description
amount	uint256	The amount to be added/subtracted from the current open interest
isDecrease	bool	True if open interest shoul be decreased. False otherwhise.

deposit

```
function deposit(uint256 amount) external payable
```

Deposits funds into the pool

Parameters

Name	Type	Description
amount	uint256	The amount of tokens to be deposited (ERC20 only)

withdraw

```
function withdraw(uint256 currencyAmount) external
```

Allows a user to withdraw his funds from the pool

Parameters

Name	Type	Description
currencyAmount	uint256	The amount of external tokens a user want to withdraw

creditUserProfit

```
function creditUserProfit(address destination, uint256 amount) external
```

Transfers currency from the pool to the given address

Parameters

Name	Type	Description
destination	address	The address to withdraw currency to
amount	uint256	The amount of currency to withdraw

fallback

```
fallback() external payable
```

Allows this contract to receive ether

receive

```
receive() external payable
```

_transferIn

```
function _transferIn(uint256 amount) internal
```

Transfers the provided amount of ERC20 tokens into the pool

Parameters

Name	Type	Description
amount	uint256	The amount of ERC20 tokens to transfer into the pool

_transferOut

```
function _transferOut(address to, uint256 amount) internal
```

Transfers currency from the pool to the given address

Parameters

Name	Type	Description
to	address	The address to withdraw currency to
amount	uint256	The amount of currency to withdraw

_getCurrentBalance

```
function _getCurrentBalance() internal view returns (uint256)
```

Returns the currency balance of the pool

Return Values

Name	Type	Description
[0]	uint256	The currency balance of the pool

getUtilization

```
function getUtilization() public view returns (uint256)
```

Returns the current utilization of the pool

Return Values

Name	Type	Description
[0]	uint256	Current utilization of the pool (in Basis Points)

getCurrencyBalance

```
function getCurrencyBalance(address account) external view returns (uint256)
```

Returns the currency balance of the given account

Parameters

Name	Type	Description
account	address	The account to get the currency balance of

Return Values

Name	Type	Description
[0]	uint256	The currency balance of the given account

getBalance

```
function getBalance(address account) external view returns (uint256)
```

Returns the PLP balance of the account

Parameters

Name	Type	Description
account	address	The account to get the PLP balance of

Return Values

Name	Type	Description
[0]	uint256	The PLP balance of the account

Name	Type	Description

onlyOwner

```
modifier onlyOwner()
```

Allows only the owner of the contract to call functions

onlyTrading

```
modifier onlyTrading()
```

Allows only the {Trading} contact to call functions

PoolParifi

owner

```
address owner
```

The address of the owner of this contract

router

```
address router
```

The address of the {Router} contract

parifi

```
address parifi
```

The address of the project token

balances

```
mapping(address => uint256) balances
```

The mapping from account's address to the amount of ERC20 tokens he deposited

totalSupply

```
uint256 totalSupply
```

The total amount of ERC20 tokens stored in the pool

DepositParifi

```
event DepositParifi(address user, uint256 amount)
```

Indicates that funds were deposited into the parifi-pool

Events

WithdrawParifi

```
event WithdrawParifi(address user, uint256 amount)
```

Indicates that funds were withdrawn from parifi-pool

constructor

```
constructor(address _parifi) public
```

setOwner

```
function setOwner(address newOwner) external
```

Seths the address of the new owner of the contract

Parameters

Name	Type	Description
newOwner	address	The address of the new owner of the contact

setRouter

```
function setRouter(address _router) external
```

Sets the address of the new router used in the contract

Parameters

Name	Type	Description
_router	address	The address of the new router

deposit

```
function deposit(uint256 amount) external
```

Allows user to deposit tokens into the pool

Parameters

Name	Type	Description
amount	uint256	The amount of tokens to be deposited (ERC20 only)

withdraw

```
function withdraw(uint256 amount) external
```

Allows a user to withdraw funds from the pool

Parameters

Name	Type	Description
amount	uint256	The amount of parifi tokens to withdraw from the pool

getBalance

```
function getBalance(address account) external view returns (uint256)
```

Returns the amount of tokens a user holds in the pool

Parameters

Name	Type	Description
account	address	The account to get the balance of

Return Values

Name	Type	Description
[0]	uint256	The amount of tokens a user holds in the pool

_update

```
function _updateRewards() internal
```

Updates rewards of the caller for each currency he staked in other pools

onlyOwner

```
modifier onlyOwner()
```

Only allows the owner of the contract to call functions

Rewards

Can either represent rewards for parifi tokens pool or rewards for currency tokens pool Each currency has at least one corresponding rewards contract

owner

```
address owner
```

The address of the owner of the contract

router

```
address router
```

The address of the {Router} contract

trading

```
address trading
```

The address of the {Trading} contract

treasury

```
address treasury
```

The address of the {Treasury} contract

pool

```
address pool
```

The address of the {Pool or PoolParifi} contract assosiated with these rewards

currency

```
address currency
```

The address of the token stored in the contract and used to pay the rewards

cumulativeRewardPerTokenStored

```
uint256 cumulativeRewardPerTokenStored
```

The reward for a single token stored in the pool by a user

pendingReward

```
uint256 pendingReward
```

The reward that has been trasfered from some other contract to this contract, but hasn't been processed in any way yet

claimableReward

```
mapping(address => uint256) claimableReward
```

Mapping from user's address to the amount of reward he can claim

previousRewardPerToken

```
mapping(address => uint256) previousRewardPerToken
```

Mapping from the user's address to the reward he claimed last time

UNIT

```
uint256 UNIT
```

Corrects decimals

CollectedReward

```
event CollectedReward(address user, address poolContract, address currency, uint256 amount)
```

Indicates that a reward has been collected by the user

constructor

```
constructor(address _pool, address _currency) public
```

setOwner

```
function setOwner(address newOwner) external
```

Sets the new owner of the contract

Parameters

Name	Type	Description
newOwner	address	The address of the new owner of the contract

setRouter

```
function setRouter(address _router) external
```

Sets the new router used in the contract

Parameters

Name	Type	Description
_router	address	The address of the new router

notifyRewardReceived

```
function notifyRewardReceived(uint256 amount) external
```

Is called by other contracts after they transfer tokens to this contract to indicate that tokens have been transferred

Parameters

Name	Type	Description
amount	uint256	The amount of tokens transferred (18 decimals)

updateRewards

```
function updateRewards(address account) public
```

Calculates the claimable reward for the account based on his pool's tokens balance and the reward for a single token

User can have either parifi or currency tokens on his balance

Parameters

Name	Type	Description
account	address	The address of account which rewards should be updated

collectReward

```
function collectReward() external
```

Allows a user to claim his reward

getClaimableReward

```
function getClaimableReward() external view returns (uint256)
```

Returns the reward amount a user can claim at the moment

Return Values

Name	Type	Description
[0]	uint256	The reward amount a user can claim at the moment

fallback

```
fallback() external payable
```

Allows this contract to receive ETH

receive

```
receive() external payable
```

_transferOut

```
function _transferOut(address to, uint256 amount) internal
```

Transfers tokens to the given address

Parameters

Name	Type	Description
to	address	The address to transfer tokens to
amount	uint256	The amount of tokens to transfer

onlyOwner

```
modifier onlyOwner()
```

Allows only the owner of the contract to call functions

onlyTreasuryOrPool

```
modifier onlyTreasuryOrPool()
```

Allows only the {Pool} or the {Treasury} contracts to call functions

Router

owner

```
address owner
```

The address of the owner of the contract

trading

```
address trading
```

The address of the {Trading} contract

oracle

```
address oracle
```

The address of the {Oracle} contract

parifiPool

```
address parifiPool
```

The address of the {PoolParifi} contract

treasury

```
address treasury
```

The address of the {Treasury} contract

darkOracle

```
address darkOracle
```

The address of the backend

factory

```
address factory
```

The address of the {Factory} contract

currencies

```
address[] currencies
```

The list of supported tokens (currencies)

decimals

```
mapping(address => uint8) decimals
```

Decimals of each of the currencies

pools

```
mapping(address => address) pools
```

The addresses of pools of each of the currencies

(currency address => pool address) Pool can either be a currency pool or a parifi tokens pool

poolShares

```
mapping(address => uint256) poolShares
```

Mapping from token address to the BPS (one hundredth of 1%) for pool share

parifiShares

```
mapping(address => uint256) parifiShares
```

Mapping from token address to the BPS (one hundredth of 1%) for parifi-pool share

poolRewards

```
mapping(address => address) poolRewards
```

Mapping from currency address to the {Rewards} contract using that currency

parifiRewards

```
mapping(address => address) parifiRewards
```

Mapping from currency address to the {Rewards} contract using that currency

constructor

```
constructor() public
```

isSupportedCurrency

```
function isSupportedCurrency(address currency) external view returns (bool)
```

Checks if the currency is supported

Parameters

Name	Type	Description
currency	address	The address of the currency to check

Return Values

Name	Type	Description
[0]	bool	True if currency is supported

currenciesLength

```
function currenciesLength() external view returns (uint256)
```

Returns the number of supported currencies

Return Values

Name	Type	Description
[0]	uint256	The number of supported currencies

getPool

```
function getPool(address currency) external view returns (address)
```

Returns the address of the pool for the given currency

Parameters

Name	Type	Description

Name	Type	Description
currency	address	The address of the currency token

Return Values

Name	Type	Description
[0]	address	The address of the pool of the currency

getPoolShare

```
function getPoolShare(address currency) external view returns (uint256)
```

Returns the pool share BPS (one hundredth of 1%) for the given currency

Parameters

Name	Type	Description
currency	address	The address of the currency

Return Values

Name	Type	Description
[0]	uint256	The pool share BPS (one hundredth of 1%)

getParifiShare

```
function getParifiShare(address currency) external view returns (uint256)
```

Returns the parifi-pool share BPS (one hundredth of 1%) for the given currency

Parameters

Name	Type	Description
currency	address	The address of the currency

Return Values

Name	Type	Description
[0]	uint256	The parifi-pool share BPS (one hundredth of 1%)

getPoolRewards

```
function getPoolRewards(address currency) external view returns (address)
```

Returns the address of the {Rewards} contract with the given currency

Parameters

Name	Type	Description
currency	address	The address of the currency

Return Values

Name	Type	Description
[0]	address	The address of the {Rewards} contract

getParifiRewards

```
function getParifiRewards(address currency) external view returns (address)
```

Returns the address of the {Rewards} contract with the given currency

Parameters

Name	Type	Description
currency	address	The address of the currency

Return Values

Name	Type	Description
[0]	address	The address of the {Rewards} contract

getDecimals

```
function getDecimals(address currency) external view returns (uint8)
```

Returns the decimals of the given currency

Parameters

Name	Type	Description
currency	address	The address of the currency to check

Return Values

Name	Type	Description
[0]	uint8	The decimals of the currency

setCurrencies

```
function setCurrencies(address[] _currencies) external
```

Sets the list of supported currencies

Parameters

Name	Type	Description
_currencies	address[]	The list of supported currencies

setDecimals

```
function setDecimals(address currency, uint8 _decimals) external
```

Sets the decimals for the given currency

Parameters

Name	Type	Description
currency	address	The address of the currency
_decimals	uint8	The decimals of the currency

setContracts

```
function setContracts(address _treasury, address _trading, address _parifiPool, address _oracle, addre
```

Sets the addresses of contracts the {Router} can call

Parameters

Name	Type	Description
_treasury	address	The address of the {Treasury} contract
_trading	address	The address of the {Trading} contract
_parifiPool	address	The address of the {PoolParifi} contract
_oracle	address	The address of the {Oracle} contract
_darkOracle	address	The address of the backend
_factory	address	The address of the {Factory} contract

setPool

```
function setPool(address currency, address _contract) external
```

Sets the pool address for the given currency

Parameters

Name	Type	Description
currency	address	The currency to set the pool for
_contract	address	The address of the pool of the currency

setPoolShare

```
function setPoolShare(address currency, uint256 share) external
```

Sets the pool share

Parameters

Name	Type	Description
currency	address	The currency of the pool to set the share for
share	uint256	The share of the pool

setParifiShare

```
function setParifiShare(address currency, uint256 share) external
```

Sets the parifi-pool share

Parameters

Name	Type	Description
currency	address	The currency of the pool to set the share for
share	uint256	The share of the parifi-pool

setPoolRewards

```
function setPoolRewards(address currency, address _contract) external
```

Sets a new {Rewards} contract for the given currency

Parameters

Name	Type	Description
currency	address	The currency to pay rewards in
_contract	address	The address of the {Rewards} contract

setParifiRewards

```
function setParifiRewards(address currency, address _contract) external
```

Sets a new {Rewards} contract for the given currency

Parameters

Name	Type	Description
currency	address	The currency to pay rewards in
_contract	address	The address of the {Rewards} contract

setOwner

```
function setOwner(address newOwner) external
```

Sets a new owner of the contract

addCurrency

```
function addCurrency(address _currency) external
```

Adds a new supported currency

Parameters

Name	Type	Description
_currency	address	The address of the currency to add

onlyOwnerOrFactory

```
modifier onlyOwnerOrFactory()
```

Allows only the owner of the contract or the {Factory} contract to call the function

Trading

Product

```
struct Product {
  uint64 maxLeverage;
  uint64 liquidationThreshold;
  uint64 fee;
  uint64 interest;
}
```

Position

```
struct Position {
  uint64 margin;
  uint64 size;
  uint64 timestamp;
  uint64 price;
  uint64 stop;
  uint64 take;
}
```

Order

```
struct Order {
  bool isClose;
  uint64 size;
  uint64 margin;
}
```

owner

```
address owner
```

The address of the owner of the contract

router

```
address router
```

The address of the {Router} contract

treasury

```
address treasury
```

The address of the {Treasury} contract

oracle

```
address oracle
```

The address of the {Oracle} contract

products

```
mapping(bytes32 => struct Trading.Product) products
```

Mapping from product IDs to products The ID of the product can be any `bytes32` value. Generally, can be generated using `keccak` over some string.

positions

```
mapping(bytes32 => struct Trading.Position) positions
```

Maping from position keys to positions Key = (currency,user,product,direction)

orders

```
mapping(bytes32 => struct Trading.Order) orders
```

*Mapping from *POSITION* keys to orders `positions` and `orders` have the same length and corresponding elements at the same indexes*

minMargin

```
mapping(address => uint256) minMargin
```

Mapping from currency to the minimum margin in that currency

pendingFees

```
mapping(address => uint256) pendingFees
```

Mapping from currency to the pending fee in that currency

UNIT_DECIMALS

```
uint256 UNIT_DECIMALS
```

In this contract the decimals of 8 is used for each token instead of 18 (like in other contracts)

UNIT

```
uint256 UNIT
```

PRICE_DECIMALS

```
uint256 PRICE_DECIMALS
```

NewOrder

```
event NewOrder(bytes32 key, address user, bytes32 productId, address currency, bool isLong, uint256 ma
```

Indicates that a new order was created

NewStopOrder

```
event NewStopOrder(bytes32 key, address user, bytes32 productId, address currency, bool isLong, uint64
```

Indicates that a new stop-loss order was created

NewTakeOrder

```
event NewTakeOrder(bytes32 key, address user, bytes32 productId, address currency, bool isLong, uint64
```

Indicates that a new take-profit order was created

PositionStopUpdated

```
event PositionStopUpdated(bytes32 key, address user, bytes32 productId, address currency, bool isLong,
```

Indicates that a stop-loss limit of the position was updated

PositionTakeUpdated

```
event PositionTakeUpdated(bytes32 key, address user, bytes32 productId, address currency, bool isLong,
```

Indicates that a take-profit limit of the position was updated

PositionUpdated

```
event PositionUpdated(bytes32 key, address user, bytes32 productId, address currency, bool isLong, uin
```

Indicates that a position was updated after settlement

ClosePosition

```
event ClosePosition(bytes32 key, address user, bytes32 productId, address currency, bool isLong, uint2
```

Indicates that a position was closed

constructor

```
constructor() public
```

setOwner

```
function setOwner(address newOwner) external
```

Sets the new owner of the contract

Parameters

Name	Type	Description
newOwner	address	The address of the new owner of the

Name	Type	contract Description

setRouter

```
function setRouter(address _router) external
```

Sets the new router used in the contract

Parameters

Name	Type	Description
_router	address	The address of the new router

setMinMargin

```
function setMinMargin(address currency, uint256 _minMargin) external
```

Sets the minimum margin for the currency

Parameters

Name	Type	Description
currency	address	The address of the currency to change the margin for
_minMargin	uint256	The new minimum margin for the currency

addProduct

```
function addProduct(bytes32 productId, struct Trading.Product _product) external
```

Adds a new product

*This function should be called to add products *before* any other functions that take `productID` as a parameter, e.g.: 1) addProduct(ID=1) 2) submitOrder(ID=1)*

Parameters

Name	Type	Description
productId	bytes32	The ID to give to a new product
_product	struct Trading.Product	The product to be added. Receives the given ID

updateProduct

```
function updateProduct(bytes32 productId, struct Trading.Product _product) external
```

Updates the product with a given ID

Parameters

Name	Type	Description
productId	bytes32	The ID of the product to update
_product	struct Trading.Product	The product that replaces the old product

distributeFees

```
function distributeFees(address currency) external
```

Distributes fees to: - Treasury contract - Pool contract (for specific currency) - Parify Pool contract (for project token staking)

submitOrder

```
function submitOrder(bytes32 productId, address currency, bool isLong, uint256 margin, uint256 size) e
```

Creates an order to open/increase a position

Parameters

Name	Type	Description
productId	bytes32	The ID of the product to use
currency	address	The currency of the position Zero address if using ether
isLong	bool	True if position is a long one (aiming for currency price increasing over time) False if position is a short one (aiming for currency price decreasing over time)
margin	uint256	The margin of the order (initial deposit)
size	uint256	The nominal amount of tokens in the order (not the same as margin)

submitCloseOrder

```
function submitCloseOrder(bytes32 productId, address currency, bool isLong, uint256 size) external pay
```

Creates an order to close/decrease a position

Parameters

Name	Type	Description
productId	bytes32	The ID of the product to use
currency	address	The currency of the position Zero address if using ether
isLong	bool	True if position is a long one (aiming for currency price increasing over time) False if position is a short one (aiming for currency price decreasing over time)
size	uint256	The nominal amount of tokens in the order (not the same as margin)

submitStopOrder

```
function submitStopOrder(bytes32 productId, address currency, bool isLong, uint64 stop) external
```

Creates an order to change a stop-loss value of the existing position

It doesn't actually create an order, but rather emits and event that imitates order creation Stop-loss limit is set for the whole position at once. It doesn't change if position gets changed.

Parameters

Name	Type	Description
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false
stop	uint64	Percent of price difference to trigger limit

submitTakeOrder

```
function submitTakeOrder(bytes32 productId, address currency, bool isLong, uint64 take) external
```

Creates an order to change a take-profit value of the existing position

It doesn't actually create an order, but rather emits and event that imitates order creation Take-profit limit is set for the whole position at once. It doesn't change if position gets changed.

Parameters

Name	Type	Description
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false

Name	Type	Description
take	uint64	Percent of price difference to trigger limit

cancelOrder

```
function cancelOrder(bytes32 productId, address currency, bool isLong) external
```

Allows user to cancel an open order

Parameters

Name	Type	Description
productId	bytes32	The ID of the product to use
currency	address	The currency of the position
isLong	bool	True if position is a long one (aiming for currency price increasing over time) False if position is a short one (aiming for currency price decreasing over time)

settleStopOrder

```
function settleStopOrder(address user, bytes32 productId, address currency, bool isLong, uint64 stop)
```

Sets stop loss for an existing position

Should be called by the backend afer {submitStopOrder}

Parameters

Name	Type	Description
user	address	The owner of position
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false
stop	uint64	Percent of price difference to trigger limit

settleTakeOrder

```
function settleTakeOrder(address user, bytes32 productId, address currency, bool isLong, uint64 take)
```

Set take profit for an existing position

Should be called by the backend afer {submitTakeOrder}

Parameters

Name	Type	Description
user	address	The owner of position
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false
take	uint64	Percent of price difference to trigger limit

settleOrder

```
function settleOrder(address user, bytes32 productId, address currency, bool isLong, uint256 price) pu
```

Sets price for a newly submitted order

Parameters

Name	Type	Description
user	address	The owner of position
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false
price	uint256	The price of the position from external source

_settleCloseOrder

```
function _settleCloseOrder(address user, bytes32 productId, address currency, bool isLong, uint256 pri
```

Settles the order for closing/decreasong the position The margin and the size of the order get returned if the position doesn't get luquidated The margin and the size of the position get returned if the position gets luquidated

Parameters

Name	Type	Description
user	address	The owner of position

Name	Type	Description
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false
price	uint256	The price of the position from external source

Return Values

Name	Type	Description
[0]	uint256	The margin (order/position), the size (order/position), PNL (positive/negative)
[1]	uint256	
[2]	int256	

settleLimit

```
function settleLimit(address user, bytes32 productId, address currency, bool isLong, uint256 price) ex
```

Closes a position by the request from the backend

Parameters

Name	Type	Description
user	address	The owner of position
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false
price	uint256	The price of the position from external source

liquidatePosition

```
function liquidatePosition(address user, bytes32 productId, address currency, bool isLong, uint256 pri
```

Liquidates a position by the request from the backend

Parameters

Name	Type	Description
user	address	The owner of position
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false
price	uint256	The price of the position from external source

releaseMargin

```
function releaseMargin(address user, bytes32 productId, address currency, bool isLong, bool includeFee
```

Transfers user's margin back to him and liquidates the position

Parameters

Name	Type	Description
user	address	The owner of position
productId	bytes32	Position's product
currency	address	Deposited token
isLong	bool	True if position is long, otherwise - false
includeFee	bool	True if fee should be released with margin, otherwise - false

fallback

```
fallback() external payable
```

These functions allow this contract to receive ether

receive

```
receive() external payable
```

_getPositionKey

```
function _getPositionKey(address user, bytes32 productId, address currency, bool isLong) internal pure
```

Hash function to get a position (order) key from multiple parameters

Parameters

Name	Type	Description
user	address	The address of the user
productId	bytes32	The ID of the product
currency	address	The address of the currency
isLong	bool	True if position is long, otherwise - false

Return Values

Name	Type	Description
[0]	bytes32	The key of the position (of the order)

_updateOpenInterest

```
function _updateOpenInterest(address currency, uint256 amount, bool isDecrease) internal
```

Updates the open interest of the pool

Parameters

Name	Type	Description
currency	address	The currency which pool should be updated
amount	uint256	The amount by which the open interest should be changed
isDecrease	bool	True if open interest should be decreased, otherwise - false

_transferIn

```
function _transferIn(address currency, uint256 amount) internal
```

Transfers currency from the caller to this contract

Parameters

Name	Type	Description
currency	address	The currency to transfer
amount	uint256	The amount of currency to transfer

_transferOut

```
function _transferOut(address currency, address to, uint256 amount) internal
```

Transfers currency from this contract to the provided address

Parameters

Name	Type	Description
currency	address	The currency to transfer
to	address	The address to transfer currency to
amount	uint256	The amount of currency to transfer

_validatePrice

```
function _validatePrice(uint256 price) internal pure returns (uint256)
```

Checks if price is valid and corrects price's decimals in necessary price The price to check (has decimals = 8)

Return Values

Name	Type	Description
[0]	uint256	A price with correct decimals

getProduct

```
function getProduct(bytes32 productId) external view returns (struct Trading.Product)
```

Returns the product with the provided ID

Parameters

Name	Type	Description
		The ID of the product to look

Name	Type	Description
productId	bytes32	for

Return Values

Name	Type	Description
[0]	struct Trading.Product	The product with the provided ID

getPosition

```
function getPosition(address user, address currency, bytes32 productId, bool isLong) external view ret
```

Returns the position with the provided ID

Parameters

Name	Type	Description
user	address	The owner of the position
currency	address	The currency of the position
productId	bytes32	The ID of the position to look for
isLong	bool	True if position is long, otherwise - false

Return Values

Name	Type	Description
position	struct Trading.Position	The position with the provided ID

getOrder

```
function getOrder(address user, address currency, bytes32 productId, bool isLong) external view return
```

Returns the order with the provided ID

Parameters

Name	Type	Description
user	address	The owner of the order

Name	Type	Description
currency	address	The currency of the order
productId	bytes32	The ID of the order to look for
isLong	bool	True if order is long, otherwise - false

Return Values

Name	Type	Description
order	struct Trading.Order	The order with the provided ID

getOrders

```
function getOrders(bytes32[] keys) external view returns (struct Trading.Order[] _orders)
```

Returns the list of orders with provided keys

Parameters

Name	Type	Description
keys	bytes32[]	The list of orders' keys

Return Values

Name	Type	Description
_orders	struct Trading.Order[]	The list of orders with provided keys

getPositions

```
function getPositions(bytes32[] keys) external view returns (struct Trading.Position[] _positions)
```

Returns the list of positions with provided keys

Parameters

Name	Type	Description
keys	bytes32[]	The list of positions' keys

Return Values

Name	Type	Description
_positions	struct Trading.Position[]	The list of positions with provided keys

getPendingFee

```
function getPendingFee(address currency) external view returns (uint256)
```

Returns the pending fee of the currency

Parameters

Name	Type	Description
currency	address	The currency of the pending fee

Return Values

Name	Type	Description
[0]	uint256	The pending fee of the currency

getPnL

```
function getPnL(bool isLong, uint256 price, uint256 positionPrice, uint256 size, uint256 interest, uin
```

Returns the PNL (profit'n'loss) of the position

Parameters

Name	Type	Description
isLong	bool	True if position is long, otherwise - false
price	uint256	The price of the position from external source
positionPrice	uint256	The price of the position from this contract
size	uint256	The nominal amount of tokens in the order (not the same as margin)
interest	uint256	The interest of the position (for 360 days)
timestamp	uint256	The time when position was settled

Return Values

Name	Type	Description
_pnl	int256	The PNL of the position

onlyOracle

```
modifier onlyOracle()
```

Allows only the {Oracle} contract to call functions Basically, the backend (a.k.a dark oracle) calls functions via {Oracle}

onlyOwner

```
modifier onlyOwner()
```

Allows only the owner of the contract to call functions

Treasury

owner

```
address owner
```

The owner of the contract

router

```
address router
```

The address of the {Router} contract

trading

```
address trading
```

The address of the {Trading} contract

oracle

```
address oracle
```

The address of the {Oracle} contract

UNIT

```
uint256 UNIT
```

constructor

```
constructor() public
```

setOwner

```
function setOwner(address newOwner) external
```

Sets the owner of the contract

Parameters

Name	Type	Description
newOwner	address	The address of the new owner of the contract

setRouter

```
function setRouter(address _router) external
```

Sets the address of the router to be used

Initialized variables with addresses received from router

Parameters

Name	Type	Description
_router	address	The new address of the router

notifyFeeReceived

```
function notifyFeeReceived(address currency, uint256 amount) external
```

Sends rewards to pool and parifi-pool contracts and notifies them about it

Parameters

Name	Type	Description
currency	address	The address of the tokens to be transferred
amount	uint256	The amount of tokens used to calculate the reward for each

Name	Type	Description
pool		

fundOracle

```
function fundOracle(address destination, uint256 amount) external
```

Sends native tokens to the oracle for its services

Parameters

Name	Type	Description
destination	address	The address of the oracle to receive funds
amount	uint256	The amount of tokens to send to the oracle

sendFunds

```
function sendFunds(address token, address destination, uint256 amount) external
```

Sends tokens from the treasury to the given address

Parameters

Name	Type	Description
token	address	The address of the token to send
destination	address	The address to transfer to
amount	uint256	The amount of tokens to transfer

fallback

```
fallback() external payable
```

Allow this contract to receive ETH

receive

```
receive() external payable
```

_transferOut

```
function _transferOut(address currency, address to, uint256 amount) internal
```

Transfers tokens to the given address

Parameters

Name	Type	Description
currency	address	The address of the token to transfer
to	address	The address to transfer tokens to
amount	uint256	The amount of tokens to transfer

onlyOwner

```
modifier onlyOwner()
```

Allows only the user of the contract to call the function

onlyTrading

```
modifier onlyTrading()
```

Allows only the {Trading} contract to call the function

onlyOracle

```
modifier onlyOracle()
```

Allows only the {Oracle} contract to call the function

IPool

setParams

```
function setParams(uint256 _minDepositTime, uint256 _utilizationMultiplier, uint256 _maxParifi, uint25
```

setRouter

```
function setRouter(address _router) external
```

totalSupply

```
function totalSupply() external view returns (uint256)
```

creditUserProfit

```
function creditUserProfit(address destination, uint256 amount) external
```

updateOpenInterest

```
function updateOpenInterest(uint256 amount, bool isDecrease) external
```

getUtilization

```
function getUtilization() external view returns (uint256)
```

getBalance

```
function getBalance(address account) external view returns (uint256)
```

IRewards

setRouter

```
function setRouter(address _router) external
```

updateRewards

```
function updateRewards(address account) external
```

notifyRewardReceived

```
function notifyRewardReceived(uint256 amount) external
```

IRouter

trading

```
function trading() external view returns (address)
```

parifiPool

```
function parifiPool() external view returns (address)
```

oracle

```
function oracle() external view returns (address)
```

treasury

```
function treasury() external view returns (address)
```

darkOracle

```
function darkOracle() external view returns (address)
```

isSupportedCurrency

```
function isSupportedCurrency(address currency) external view returns (bool)
```

currencies

```
function currencies(uint256 index) external view returns (address)
```

currenciesLength

```
function currenciesLength() external view returns (uint256)
```

getDecimals

```
function getDecimals(address currency) external view returns (uint8)
```

getPool

```
function getPool(address currency) external view returns (address)
```

getPoolShare

```
function getPoolShare(address currency) external view returns (uint256)
```

getParifiShare

```
function getParifiShare(address currency) external view returns (uint256)
```

getPoolRewards

```
function getPoolRewards(address currency) external view returns (address)
```

getParifiRewards

```
function getParifiRewards(address currency) external view returns (address)
```

setPool

```
function setPool(address currency, address _contract) external
```

setPoolRewards

```
function setPoolRewards(address currency, address _contract) external
```

setParifiRewards

```
function setParifiRewards(address currency, address _contract) external
```

setCurrencies

```
function setCurrencies(address[] _currencies) external
```

setDecimals

```
function setDecimals(address currency, uint8 _decimals) external
```

setPoolShare

```
function setPoolShare(address currency, uint256 share) external
```

setParifiShare

```
function setParifiShare(address currency, uint256 share) external
```

addCurrency

```
function addCurrency(address _currency) external
```

ITrading

distributeFees

```
function distributeFees(address currency) external
```

settleOrder

```
function settleOrder(address user, bytes32 productId, address currency, bool isLong, uint256 price) ex
```

settleLimit

```
function settleLimit(address user, bytes32 productId, address currency, bool isLong, uint256 price) ex
```

liquidatePosition

```
function liquidatePosition(address user, bytes32 productId, address currency, bool isLong, uint256 pri
```

getPendingFee

```
function getPendingFee(address currency) external view returns (uint256)
```

settleStopOrder

```
function settleStopOrder(address user, bytes32 productId, address currency, bool isLong, uint64 stop)
```

settleTakeOrder

```
function settleTakeOrder(address user, bytes32 productId, address currency, bool isLong, uint64 take)
```

ITreasury

fundOracle

```
function fundOracle(address destination, uint256 amount) external
```

notifyFeeReceived

```
function notifyFeeReceived(address currency, uint256 amount) external
```

Address

Collection of functions related to the address type

isContract

```
function isContract(address account) internal view returns (bool)
```

_Returns true if `account` is a contract.

[IMPORTANT]

It is unsafe to assume that an address for which this function returns false is an externally-owned account (EOA) and not a contract.

Among others, `isContract` will return false for the following types of addresses:

- an externally-owned account
- a contract in construction
- an address where a contract will be created
- an address where a contract lived, but was destroyed ====_

sendValue

```
function sendValue(address payable recipient, uint256 amount) internal
```

_Replacement for Solidity's `transfer`: sends `amount` wei to `recipient`, forwarding all available gas and reverting on errors.

https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost of certain opcodes, possibly making contracts go over the 2300 gas limit imposed by `transfer`, making them unable to receive funds via `transfer`. {sendValue} removes this limitation.

https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].

IMPORTANT: because control is transferred to `recipient`, care must be taken to not create reentrancy vulnerabilities. Consider using {ReentrancyGuard} or the https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern]._

functionCall

```
function functionCall(address target, bytes data) internal returns (bytes)
```

_Performs a Solidity function call using a low level `call`. A plain `call` is an unsafe replacement for a function call: use this function instead.

If `target` reverts with a revert reason, it is bubbled up by this function (like regular Solidity function calls).

Returns the raw returned data. To convert to the expected return value, use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].

Requirements:

- `target` must be a contract.
- calling `target` with `data` must not revert.

_Available since v3.1.__

functionCall

```
function functionCall(address target, bytes data, string errorMessage) internal returns (bytes)
```

_Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with `errorMessage` as a fallback revert reason when `target` reverts.

_Available since v3.1.__

functionCallWithValue

```
function functionCallWithValue(address target, bytes data, uint256 value) internal returns (bytes)
```

_Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but also transferring `value` wei to `target`.

Requirements:

- the calling contract must have an ETH balance of at least `value`.
- the called Solidity function must be `payable`.

_Available since v3.1.__

functionCallWithValue

```
function functionCallWithValue(address target, bytes data, uint256 value, string errorMessage) interna
```

_Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but with `errorMessage` as a fallback revert reason when `target` reverts.

_Available since v3.1.__

functionStaticCall

```
function functionStaticCall(address target, bytes data) internal view returns (bytes)
```

_Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but performing a static call.

_Available since v3.3.__

functionStaticCall

```
function functionStaticCall(address target, bytes data, string errorMessage) internal view returns (by
```

_Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`], but performing a static call.

_Available since v3.3.__

functionDelegateCall

```
function functionDelegateCall(address target, bytes data) internal returns (bytes)
```

_Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but performing a delegate call.

_Available since v3.4.__

functionDelegateCall

```
function functionDelegateCall(address target, bytes data, string errorMessage) internal returns (bytes
```

_Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`], but performing a delegate call.

_Available since v3.4.__

verifyCallResult

```
function verifyCallResult(bool success, bytes returndata, string errorMessage) internal pure returns (
```

_Tool to verifies that a low level call was successful, and revert if it wasn't, either by bubbling the revert reason using the provided one.

_Available since v4.3.__

SafeERC20

Wrappers around ERC20 operations that throw on failure (when the token contract returns false). Tokens that return no value (and instead revert or throw on failure) are also supported, non-reverting calls are assumed to be successful. To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract, which allows you to call the safe operations as `token.safeTransfer(...)`, etc.

safeTransfer

```
function safeTransfer(contract IERC20 token, address to, uint256 value) internal
```

safeTransferFrom

```
function safeTransferFrom(contract IERC20 token, address from, address to, uint256 value) internal
```

safeApprove

```
function safeApprove(contract IERC20 token, address spender, uint256 value) internal
```

_Deprecated. This function has issues similar to the ones found in {IERC20-approve}, and its usage is discouraged.

Whenever possible, use {safeIncreaseAllowance} and {safeDecreaseAllowance} instead._

safeIncreaseAllowance

```
function safeIncreaseAllowance(contract IERC20 token, address spender, uint256 value) internal
```

safeDecreaseAllowance

```
function safeDecreaseAllowance(contract IERC20 token, address spender, uint256 value) internal
```

_callOptionalReturn

```
function _callOptionalReturn(contract IERC20 token, bytes data) private
```

Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement on the return value: the return value is optional (but if data is returned, it must not be false).

Parameters

Name	Type	Description
token	contract IERC20	The token targeted by the call.
data	bytes	The call data (encoded using abi.encode or one of its variants).

MockToken

_decimals

```
uint8 _decimals
```

constructor

```
constructor(string name, string symbol, uint8 __decimals) public
```

decimals

```
function decimals() public view virtual returns (uint8)
```

_Returns the number of decimals used to get its user representation. For example, if `decimals` equals `2`, a balance of `505` tokens should be displayed to a user as `5.05` (`505 / 10 ** 2`).

Tokens usually opt for a value of 18, imitating the relationship between Ether and Wei. This is the value {ERC20} uses, unless this function is overridden;

NOTE: This information is only used for *display* purposes: it in no way affects any of the arithmetic of the contract, including {IERC20-balanceOf} and {IERC20-transfer}._

mint

```
function mint(uint256 amount) public
```

IOracle