Lido

- Source code
- Deployed contract

Liquid staking pool and a relatedERC-20 rebasing token (stETH)

What is Lido?

Lido is a liquid staking pool and the core contract that is responsible for:

- accepting users' stake, buffering it and minting respective amounts of liquid token
- do a proper accounting based on received oracle reports and the current state of the protocol
- collecting withdrawals, priority fees and MEV from respective vaults into the buffer
- applying fees and distributing rewards
- passing buffered ether further toStakingRouter
- orWithdrawalQueueERC721

Also, Lido is anERC-20 rebasing token, which represents staked ether,stETH . Tokens are minted upon ether submission and burned when redeemed.stETH holder balances are updated daily with oracle reports. It also implements theERC-2612 permit andERC-1271 signature validation extensions.

Other contracts are bound to the core and have the following responsibilities:

- LidoLocator
- : protocol-wide address book which contains
- references to all meaningful parts of the Lido protocol on-chain
- WithdrawalQueueERC721
- : a withdrawal requests
- FIFO queue and a respective NFT (unstETH)
- StakingRouter
- : hub, which manages staking modules and
- distributes the stake among them
- NodeOperatorsRegistry
- : original module,
- responsible for managing the curated set of node operators
- OracleReportSanityChecker
- : helper for
- validation of oracle report parameters and smoothening token rebases
- Burner
- : vault to containstETH
- that ought to be burned on
- oracle report
- WithdrawalVault
- : vault to collect partial and full
- withdrawals coming from the Beacon Chain
- LidoExecutionLayerRewardsVault
- :
- vault to collect priority fees and MEV rewards coming from validators of the pool
- DepositSecurityModule
- : protection from
- deposit frontrunning vulnerability
- AccountingOracle
- : oracle committee, which gathers
- an accounting report for the protocol
- EIP712StETH
- : ad-hoc helper to implement ERC-2612 permit
- for Solidity 0.4.24 Lido contract

Submit

Lido contract is a main entry point for stakers. To take part in the pool, a user can send some ETH to the contract address and the same amount ofstETH tokens will be minted to the sender address. Submitted ether is accumulated in the buffer and can be passed further toWithdrawalQueueERC721 to fulfill withdrawal requests or toStakingRouter to deposit as a new validator stake.

To withdraw the underlying ETH back, a user may use theWithdrawalQueueERC721 contract or swap the token on the secondary market (it may be a cheaper and faster alternative).

Deposit

User-submitted ether is stored in the buffer and can be later used for withdrawals or passed further to StakingRouter to be used as validator's deposits. It happens asynchronously and uses DepositSecurityModule as a guard to prevent deposit frontrunning vulnerability.

Redeem

The token might be redeemed for ether through the protocol using the WithdrawalQueueERC721 contract leveraging staking withdrawals enabled with the Shanghai/Capella (aka "Shapella") Ethereum hardfork.

Rebase

When an oracle report occurs, the supply of the token is increased or decreased algorithmically, based on staking rewards (or slashing penalties) on the Beacon Chain, execution layer rewards (starting from the Merge Ethereum upgrade) or fulfilled withdrawal requests (starting from Lido V2). A rebase happens when oracle reports beacon stats.

The rebasing mechanism is implemented via the "shares" concept. Instead of storing a map with account balances, Lido stores which share of the total pool is owned by the account. The balance of an account is calculated as follows:

balanceOf (account)

= shares [account]

* totalPooledEther / totalShares * shares * - map of user account shares. Every time a user deposits ether, it is * converted to shares and added to the current user shares amount. * totalShares * - the sum of shares of all accounts in theshares * map * totalPooledEther * - a sum of three types of ether owned by protocol: * * buffered balance - ether stored on contract and hasn't been deposited or * * locked for withdrawals yet * * transient balance - ether submitted to the official Deposit contract but not * * yet visible in the beacon state * * beacon balance - the total amount of ether on validator accounts. This value * * is reported by oracles and makes the strongest impact on stETH total supply * * change

For example, assume that we have:

totalShares

5 totalPooledEther =

10

ETH sharesOf (Alice)

-

1 sharesOf (Bob)

-

4 Therefore:

balanceOf (Alice)

-

2 tokens which corresponds to 2

ETH balanceOf (Bob)

-

8 tokens which corresponds to 8

ETH On each rebasetotalPooledEther normally increases, indicating that there were some rewards earned by validators, that ought to be distributed, so the user balance gets increased as well automatically, despite their shares remaining as they were.

totalPooledEther

15

ETH // user balance increased balanceOf (Alice)

-

3 tokens which corresponds to 3

ETH now balanceOf (Bob)

-

12 tokens which corresponds to 12

ETH now // shares remain still sharesOf (Alice)

-

1 sharesOf (Bob)

-

4 note Since the balances of all token holders change when the amount of total pooled ether changes, this token cannot fully implement the ERC-20 standard: it only emitsTransfer events upon explicit transfer between holders. In contrast, when the total amount of pooled ether increases, noTransfer events are generated: doing so would require emitting an event for each token holder and thus running an unbounded loop.

Oracle report

One of the cornerstones of the Lido protocol is the oracle report, that usually (but not guaranteed) once a day provides the protocol with the data that can't be easily accessed on-chain, but is required for precise accounting. It includes some Beacon chain stats as well as corresponding EL-side values that are valid on the reporting block and the decision data required to fulfill pending withdrawal requests.

- Beacon chain stats:* the total number of validators managed by the pool
-
    - the total balance of validators managed by the pool
- Historical EL values:* withdrawal vault balance
-
    - execution layer rewards vault balance
-
    - burner stETH shares balance
- Withdrawal-related data* requests in the queue to be finalized
-
    - share rate to be used for finalization

Oracle report is processed in 9 simple steps:

1. Memorize the pre-state that will be required for incremental updates of the
2. protocol balance
3. Validate the report data usingOracleReportSanityChecker
4. Calculate the amount of ether to be locked onWithdrawalQueueERC721
5. and move the respective amount of shares to be burnt toBurner
6. UsingOracleReportSanityChecker
7. calculate the amounts of ether that can be withdrawn fromLidoExecutionLayerRewardsVault
8. andWithdrawalVault
9. as well as the number of shares that
10. can be burnt fromBurner
11. to avoid the rebase that can be easily
12. frontrun.
13. Collect the calculated amounts of ether from vaults and proceed with
14. withdrawal requests finalization: send requested ether toWithdrawalQueue
15. Burn the previously requested shares fromBurner
16. for
17. withdrawals or coverage application
18. Distribute rewards and protocol fees minting new stETH for the respective
19. parties
20. Complete token rebase by informing observers (emit an event and call the
21. external receivers if any)

22. Post-report sanity check for share rate provided with the report

So, the observable outcome of the report for the protocol is the following:

- withdrawal requests in the queue are fulfilled
- ether is collected from withdrawal and EL rewards vaults to the buffer
- CL balance is updated according to the report
- rewards are distributed among stakers, staking modules and protocol treasury

Standards

Contract implements the following Ethereum standards:

- ERC-20: Token Standard
- ERC-2612: Permit Extension for ERC-20 Signed Approvals
- EIP-712: Typed structured data hashing and signing
- ERC-1271: Standard Signature Validation Method for Contracts

Staking-related Methods

fallback

Sends funds to the pool and mintsStETH tokens to themsg.sender address

function ( )

payable note Allows users to submit their funds by sending it to the contract address

submit()

Sends funds to the pool with the optional_referral parameter and mintsStETH tokens to themsg.sender address.

Seehttps://lido.fi/referral for referral program details.

function

submit (address _referral)

payable

returns

(uint256) Parameter Type Description _referral address Optional referral address Returns the number ofStETH shares generated.

getBufferedEther()

Returns the amount of ether temporarily buffered on the contract's balance.

function

getBufferedEther ( )

view

returns

(uint256) note The buffered balance is kept on the contract from the moment the funds are received from a user until the moment they are sent to the officialDeposit contract orWithdrawalsQueueERC721

isStakingPaused()

Returns staking state: whether it's paused or not.

function

isStakingPaused ( )

view

returns

(bool) note 'staking' here means the ability to accept newsubmit requests

getCurrentStakeLimit()

Returns how much ether can be staked in the current block.

function

getCurrentStakeLimit ( )

view

returns

(uint256) note Special return values:

- 2^256 - 1
- if staking is unlimited;
- 0
- if staking is paused or if the limit is exhausted.

getStakeLimitFullInfo()

Returns full info about current stake limit parameters and state.

function

getStakeLimitFullInfo ( )

view

returns

(bool isStakingPaused , bool isStakingLimitSet , uint256 currentStakeLimit , uint256 maxStakeLimit , uint256 maxStakeLimitGrowthBlocks , uint256 prevStakeLimit , uint256 prevStakeBlockNumber) Name Type Description isStakingPaused bool Staking pause state (equivalent to return ofisStakingPaused()) isStakingLimitSet bool Whether the stake limit is set or not currentStakeLimit uint256 Current stake limit (equivalent to return ofgetCurrentStakeLimit()) maxStakeLimit uint256 Max stake limit maxStakeLimitGrowthBlocks uint256 Blocks needed to restore max stake limit from the fully exhausted state prevStakeLimit uint256 Previously reached stake limit prevStakeBlockNumber uint256 Previously seen block number

Deposit-related methods

deposit()

Deposit buffered ether to the StakingRouter's module with the id of_stakingModuleId .

Can be called only byDepositSecurityModule contract.

function

deposit (uint256 _maxDeposits ,

uint256 _stakingModuleId ,

bytes _depositCalldata) Parameter Type Description _maxDeposits uint256 Number of max deposit calls _stakingModuleId uint256 Id of the staking module to be deposited _depositCalldata bytes module calldata

getDepositableEther()

Returns the amount of ether available to deposit.

function

getDepositableEther ( )

view

returns

(uint256)

canDeposit()

Returnstrue if depositing buffered ether to the consensus layer is allowed.

function

canDeposit ( )

view

returns

(bool)

Accounting-related methods

handleOracleReport()

Updates accounting stats, collects EL rewards and distributes collected rewards if beacon balance increased, performs withdrawal requests finalization.

Can be called only byAccountingOracle contract.

function

handleOracleReport (uint256 _reportTimestamp , uint256 _timeElapsed , uint256 _clValidators , uint256 _clBalance , uint256 _withdrawalVaultBalance , uint256 _elRewardsVaultBalance , uint256 _sharesRequestedToBurn , uint256 [ ] _withdrawalFinalizationBatches , uint256 _simulatedShareRate)

returns

(uint256 [4] postRebaseAmounts) Parameter Type Description _reportTimestamp uint256 The moment of the oracle report calculation _timeElapsed uint256 Seconds elapsed since the previous report calculation _clValidators uint256 Number of Lido validators on Consensus Layer _clBalance uint256 Sum of all Lido validators' balances on Consensus Layer _withdrawalVaultBalance uint256 Withdrawal vault balance on Execution Layer _elRewardsVaultBalance uint256 elRewards vault balance on Execution Layer _sharesRequestedToBurn uint256 shares requested to burn through Burner _withdrawalFinalizationBatches uint256[] the ascendingly-sorted array of withdrawal request IDs to finalize _simulatedShareRate uint256 share rate simulated by oracle when (1e27 precision) Returns a fixed array of 4 values that represents changes made during the report

Name Type Description postRebaseAmounts[0] uint256 postTotalPooledEther amount of ether in the protocol postRebaseAmounts[1] uint256 postTotalShares amount of shares in the protocol postRebaseAmounts[2] uint256 withdrawals withdrawn from the withdrawals vault postRebaseAmounts[3] uint256 elRewards withdrawn from the execution layer rewards vault

getTotalPooledEther()

Returns the entire amount of ether controlled by the protocol

function

getTotalPooledEther ( )

view

returns

(uint256) note The sum of all ETH balances in the protocol, equals to the total supply ofstETH .

getTotalELRewardsCollected()

Returns the total amount of execution layer rewards collected to Lido contract buffer.

function

getTotalELRewardsCollected ( )

view

returns

(uint256)

getBeaconStat()

Returns the tuple of key statistics related to the Beacon Chain.

function

getBeaconStat ( )

view

returns

(uint256 depositedValidators , uint256 beaconValidators , uint256 beaconBalance) Name Type Description depositedValidators uint256 Number of the ever deposited Lido-participating validators beaconValidators uint256 Number of Lido's validators visible in the Beacon state, reported by oracles beaconBalance uint256 Total amount of Beacon-side ether (sum of all the balances of Lido validators) note depositedValidators is always greater or equal tobeaconValidators |

receiveELRewards()

A payable function for execution layer rewards. Can be called only by the LidoExecutionLayerRewardsVault contract.

function

receiveELRewards ( )

payable

receiveWithdrawals()

A payable function for withdrawals acquisition. Can be called only by WithdrawalVault .

function

receiveWithdrawals ( )

payable

Protocol levers

stop()

Stop pool routine operations.

Can be called only by the bearer ofPAUSE_ROLE

function

stop ( )

resume()

Resume pool routine operations.

Can be called only by the bearer ofRESUME_ROLE

function

resume ( )

pauseStaking()

Stops accepting new ether to the protocol.

Can be called only by the bearer ofSTAKING_PAUSE_ROLE

function

pauseStaking ( ) note While accepting new ether is stopped, calls to thesubmit function, as well as to the default payable function, will revert.

resumeStaking()

Resumes accepting new ether to the protocol (ifpauseStaking was called previously).

Can be called only by the bearer ofSTAKING_CONTROL_ROLE

function

resumeStaking ( ) note Staking could be rate-limited by imposing a limit on the stake amount at each moment in time, see[setStakingLimit()]() and[removeStakingLimit()]() .

setStakingLimit()

Sets the staking rate limit.

Can be called only by the bearer ofSTAKING_CONTROL_ROLE

Limit explanation scheme:

- ▲ Stake limit
- │ .... Stake limit = max
- │ . . . .
- │ . . . .
- │ . . . . .
- │ ——> Time
- │ ^ ^ ^ ^^^ ^ ^ ^ ^^^ ^ Stake events function

setStakingLimit (uint256 _maxStakeLimit ,

uint256 _stakeLimitIncreasePerBlock) Parameter Type Description _maxStakeLimit uint256 Max stake limit value _stakeLimitIncreasePerBlock uint256 Stake limit increase per single block note Reverts if:

- _maxStakeLimit
- == 0
- _maxStakeLimit

- = 2^96

- _maxStakeLimit
- <_stakeLimitIncreasePerBlock
- _maxStakeLimit
- /_stakeLimitIncreasePerBlock

- = 2^32 (only if_stakeLimitIncreasePerBlock

- != 0)

removeStakingLimit()

Removes the staking rate limit.

Can be called only by the bearer ofSTAKING_CONTROL_ROLE

function

removeStakingLimit ( )

unsafeChangeDepositedValidators()

Unsafely change deposited validators counter.

The method unsafely changes the deposited validator counter. Can be required when onboarding external validators to Lido (i.e., had deposited before and rotated their type-0x00 withdrawal credentials to Lido).

Can be called only by the bearer ofUNSAFE_CHANGE_DEPOSITED_VALIDATORS_ROLE

Parameter Type Description _newDepositedValidators uint256 New value for the deposited validators counter warning The method might break the internal protocol state if applied incorrectly

ERC-20

-related Methods

name()

Returns the name of the token.

function

name ( )

view

returns

(string) note Always returnsLiquid staked Ether 2.0 .

symbol()

Returns the symbol of the token.

function

symbol ( )

view

returns

(string) note Always returnsstETH .

decimals()

Returns the number of decimals for getting user representation of a token amount.

function

decimals ( )

view

returns

(uint8) note Always returns18 .

totalSupply()

Returns the number of tokens in existence.

function

totalSupply ( )

view

returns

(uint256) note Always equals togetTotalPooledEther() since the token amount is pegged to the total amount of ether controlled by the protocol.

balanceOf()

Returns the number of tokens owned by the_account

function

balanceOf (address _account)

view

returns

(uint256) note Balances are dynamic and equal the_account 's share in the amount of total ether controlled by the protocol. SeesharesOf .

allowance()

Returns the remaining number of tokens that_spender is allowed to spend on behalf of_owner throughtransferFrom() . This is zero by default.

function

allowance (address _owner ,

address _spender)

view

returns

(uint256) Parameter Type Description _owner address Address of owner _spender address Address of spender note This value changes whenapprove() ortransferFrom() is called unless the allowance is infinite (2^256)

approve()

Sets_amount as the allowance of_spender over the caller's tokens

function

approve (address _spender ,

uint256 _amount)

returns

(bool) Parameter Type Description _spender address Address of spender _amount uint256 Amount of tokens Returns a boolean value indicating whether the operation succeeded.

note Requirements:

- _spender
- cannot be the zero address.
- the contract must not be paused.

increaseAllowance()

Atomically increases the allowance granted to_spender by the caller by_addedValue .

This is an alternative toapprove() that can be used as a mitigation for problems describedhere .

function

increaseAllowance (address _spender ,

uint256 _addedValue)

returns

(bool) Parameter Type Description _sender address Address of spender _addedValue uint256 Amount of tokens to increase allowance Returns a boolean value indicating whether the operation succeeded

note Requirements:

- _spender
- address cannot be zero.
- the contract must not be paused.

decreaseAllowance()

Atomically decreases the allowance granted to_spender by the caller by_subtractedValue .

This is an alternative toapprove() that can be used as a mitigation for problems describedhere .

function

decreaseAllowance (address _spender ,

uint256 _subtractedValue)

returns

(bool) Parameter Type Description _sender address Address of spender _subtractedValue uint256 Amount of tokens to decrease allowance Returns a boolean value indicating whether the operation succeeded.

note Requirements:

- _spender
- cannot be the zero address.
- _spender
- must have an allowance for the caller of at least_subtractedValue
- .
- the contract must not be paused.

transfer()

Moves_amount tokens from the caller's account to the_recipient account.

function

transfer (address _recipient ,

uint256 _amount)

returns

(bool) Parameter Type Description _recipient address Address of tokens recipient _amount uint256 Amount of tokens to transfer Returns a boolean value indicating whether the operation succeeded.

note Requirements:

- _recipient
- cannot be the zero address or stETH contract itself.
- the caller must have a balance of at least_amount
- .
- the contract must not be paused.

transferFrom()

Moves_amount tokens from_sender to_recipient using the allowance mechanism._amount is then deducted from the caller's allowance.

function

transferFrom (address _sender , address _recipient , uint256 _amount)

returns

(bool) Parameter Type Description _sender address Address of spender _recipient address Address of recipient _amount uint256 Amount of tokens Returns a boolean value indicating whether the operation succeeded.

note Requirements:

- _sender
- cannot be the zero addresses.
- _recipient
- cannot be the zero address or stETH contract itself.
- _sender
- must have a balance of at least_amount
- .
- the caller must have an allowance for_sender
- 's tokens of at least_amount
- .

- the contract must not be paused.

Shares-related Methods

getTotalShares()

Returns the total amount of shares in existence.

function

getTotalShares ( )

view

returns

(uint256)

sharesOf()

Returns the number of shares owned by_account

function

sharesOf (address _account)

view

returns

(uint256)

getSharesByPooledEth()

Returns the number of shares that corresponds to_ethAmount of the protocol-controlled ether.

function

getSharesByPooledEth (uint256 _ethAmount)

view

returns

(uint256)

getPooledEthByShares()

Returns the amount of ether that corresponds to_sharesAmount token shares.

function

getPooledEthByShares (uint256 _sharesAmount)

view

returns

(uint256)

transferShares()

Moves token shares from the caller's account to the provided recipient account.

function

transferShares (address _recipient ,

uint256 _sharesAmount)

returns

(uint256) Parameter Type Description _recipient address Address of shares recipient _sharesAmount uint256 Amount of shares to transfer Returns the number Amount of transferred tokens.

note Requirements:

- _recipient
- cannot be the zero address or stETH contract itself.
- the caller must have at least_sharesAmount
- shares.
- the contract must not be paused.

transferSharesFrom()

Moves_sharesAmount token shares from the_sender account to the_recipient using the allowance mechanism. The amount of tokens equivalent to_sharesAmount is then deducted from the caller's allowance.

function

transferSharesFrom (address _sender , address _recipient , uint256 _sharesAmount)

returns

(uint256) Parameter Type Description _sender address Address of spender _recipient address Address of recipient _sharesAmount uint256 Amount of shares Returns the number of transferred tokens.

note Requirements:

- _sender
- cannot be the zero address.
- _recipient
- cannot be the zero address or stETH contract itself.
- _sender
- must have at least_sharesAmount
- shares.
- the caller must have an allowance for_sender
- 's tokens of at leastgetPooledEthByShares(_sharesAmount)
- .
- the contract must not be paused.

ERC-2612

-related methods

nonces()

Returns the current nonce for theowner . This value must be included whenever a signature is generated for an ERC-2612 permit.

function

nonces (address owner)

view

returns

(uint256)

DOMAIN_SEPARATOR()

Returns the domain separator used in the encoding of the signature for the ERC-2612 permit, as defined by EIP-712.

function

DOMAIN_SEPARATOR ( )

view

returns

(bytes32)

permit()

Setsvalue as the allowance ofspender overowner 's tokens, givenowner 's signed approval.

Emits an Approval event.

function

permit (address owner ,

address spender ,

uint256 value ,

uint256 deadline ,

uint8 v ,

bytes32 r ,

bytes32 s) Parameter Type Description owner address Address of spender spender address Address of recipient value uint256 Allowance to set deadline uint256 Permit expiry time v uint8 extendedsecp256k1 signature r bytes32 extendedsecp256k1 signature s bytes32 extendedsecp256k1 signature note Requirements:

- spender
- cannot be the zero address.
- deadline
- must be a timestamp in the future.
- v
- ,r
- ands
- must be a validsecp256k1
- signature fromowner
- over the EIP712-formatted function arguments.
- the signature must useowner
- 's current nonce (seenonces
-).

ERC-712

-related methods

eip712Domain()

Returns the fields and values that describe the domain separator used by this contract for EIP-712

function

eip712Domain ( )

view

returns

(string name , string version , uint256 chainId , address verifyingContract)

General Methods

getLidoLocator()

Returns the address ofLidoLocator .

function

getLidoLocator ( )

view

returns

(address)

getContractVersion()

Returns the current contract version.

function

getContractVersion ( )

view

returns

(uint256) note Always returns2 .

transferToVault()

Overrides default AragonApp behavior to disallow recovery.

function

transferToVault (address _token) Parameter Type Description _token address Token to be sent to recovery vault note Always reverts withNOT_SUPPORTED reason Edit this page Previous LidoLocator Next EIP712StETH