

# OracleReportSanityChecker

- [Source code](#)
- [Deployed contract](#)

Some vital data for the Lido protocol is collected off-chain and delivered on-chain via Oracle contracts [AccountingOracle](#), [ValidatorsExitBusOracle](#). Due to the high impact of data provided by the Oracles on the state of the protocol, each Oracle's report passes a set of onchain [sanity checks](#). For the simplicity of the contracts responsible for handling Oracle's reports, all sanity checks were collected in the standalone [OracleReportSanityChecker](#) contract.

Besides the validation methods, the [OracleReportSanityChecker](#) contract contains [a set of tunable limits and restrictions](#) used during the report validation process. To configure the limits values contract provides the lever methods described in the [standalone section](#). Access to lever methods is restricted using the functionality of the [AccessControlEnumerable](#) contract and a bunch of [granular roles](#).

## Limits List

[OracleReportSanityChecker](#) introduces a new type [LimitsList](#) which contains all the limits used by the contract.

struct

LimitsList

```
{ uint256 churnValidatorsPerDayLimit ; uint256 oneOffCLBalanceDecreaseBPLimit ; uint256 annualBalanceIncreaseBPLimit ; uint256 simulatedShareRateDeviationBPLimit ; uint256 maxValidatorExitRequestsPerReport ; uint256 maxAccountingExtraDataListItemsCount ; uint256 maxNodeOperatorsPerExtraDataItemCount ; uint256 requestTimestampMargin ; uint256 maxPositiveTokenRebase ; }
```

\* churnValidatorsPerDayLimit \*  $\in [0, 65535]$  \* — the max possible number of validators that might been reported as appeared \* or exited \* during a single day. [AccountingOracle](#) \* reports validators as appeared \* once they become pending \* (might be not activated \* yet). Thus, this limit should be high enough for such cases because Consensus Layer has no \* intrinsic churn limit for the amount of pending \* validators (only for activated \* instead). \* For Lido it's limited by the max daily deposits via [DepositSecurityModule](#) \*. In contrast, exited \* are reported according to the [Consensus Layer churn limit](#) \*. \* oneOffCLBalanceDecreaseBPLimit \*  $\in [0, 10000]$  \* — the max decrease of the total validators' balances on the Consensus Layer since \* the previous oracle report. Represented in the [Basis Points](#) \* (100% == 10000). \* annualBalanceIncreaseBPLimit \*  $\in [0, 10000]$  \* — the max annual increase of the total validators' balances on the Consensus Layer \* since the previous oracle report. Represented in the [Basis Points](#) \* (100% == 10000). \* simulatedShareRateDeviationBPLimit \*  $\in [0, 10000]$  \* — the max deviation of the provided simulatedShareRate \* and the actual one within the \* currently processing oracle report. Represented in the [Basis Points](#) \* (100% == 10000). \* maxValidatorExitRequestsPerReport \*  $\in [0, 65535]$  \* — the max number of exit requests allowed in report \* to [ValidatorsExitBusOracle](#) \* maxAccountingExtraDataListItemsCount \*  $\in [0, 65535]$  \* — the max number of data list items reported to accounting oracle in extra data \* maxNodeOperatorsPerExtraDataItemCount \*  $\in [0, 65535]$  \* — the max number of node operators reported per extra data list item \* requestTimestampMargin \*  $\in [0, \text{type(uint64).max}]$  \* — the min time required to be passed from the creation of the request to be finalized till the time of the oracle report \* maxPositiveTokenRebase \*  $\in [1, \text{type(uint64).max}]$  \* — the max positive token rebase allowed per single oracle report token rebase \* happens on total supply adjustment, huge positive rebase can incur oracle report sandwiching. \* Uses 1e9 precision, e.g.: 1e6 \* — 0.1%; 1e9 \* — 100%; type(uint64).max \* — unlimited rebase.

## Sanity Checks

### checkAccountingOracleReport()

Applies sanity checks to the accounting parameters of Lido's Oracle report.

note Below is the list of restrictions checked by the method execution:

- Revert with IncorrectWithdrawalsVaultBalance(uint256 actualWithdrawalVaultBalance)
- error when the reported withdrawals
- vault balance is greater than
- the actual balance of the withdrawal vault.
- Revert with IncorrectELRewardsVaultBalance(uint256 actualELRewardsVaultBalance)
- error when reported EL rewards vault
- balance is greater than
- the actual balance of EL rewards vault.
- Revert with IncorrectSharesRequestedToBurn(uint256 actualSharesToBurn)
- error when the amount of stETH shares requested
- to burn exceeds
- the number of shares marked to be burned in the Burner contract.
- Revert with IncorrectCLBalanceDecrease(uint256 oneOffCLBalanceDecreaseBP)

- error when Consensus Layer one-off balance
- decrease in basis pointsexceeds
- the allowedLimitsList.oneOffCLBalanceDecreaseBPLimit
- .
- Revert withIncorrectCLBalanceIncrease(uint256 annualBalanceDiff)
- error when Consensus Layer annual balance increase
- expressed in basis pointsexceeds
- allowedLimitsList.annualBalanceIncreaseBPLimit
- .
- Revert withIncorrectAppearedValidators(uint256 churnLimit)
- error when the number of appeared validatorsexceeds
- the limit set byLimitsList.churnValidatorsPerDayLimit
- . function

checkAccountingOracleReport ( uint256 \_timeElapsed , uint256 \_preCLBalance , uint256 \_postCLBalance , uint256 \_withdrawalVaultBalance , uint256 \_elRewardsVaultBalance , uint256 \_sharesRequestedToBurn , uint256 \_preCLValidators , uint256 \_postCLValidators )

### Arguments

- \_timeElapsed
- — time elapsed since the previous oracle report, measured inseconds
- \_preCLBalance
- — sum of all Lido validators' balances on the Consensus Layer before the current oracle report
- (NB: also include the initial balance of newly appeared validators)
- \_postCLBalance
- — sum of all Lido validators' balances on the Consensus Layer after the current oracle report
- \_withdrawalVaultBalance
- — withdrawal vault balance on Execution Layer for the report reference slot
- \_elRewardsVaultBalance
- — el rewards vault balance on Execution Layer for the report reference slot
- \_sharesRequestedToBurn
- — shares requested to burn for the report reference slot
- \_preCLValidators
- — Lido-participating validators on the CL side before the current oracle report
- \_postCLValidators
- — Lido-participating validators on the CL side after the current oracle report

### checkExitBusOracleReport()

Validates that number of exit requests does not exceed the limit set byLimitsList.maxValidatorExitRequestsPerReport .

note Reverts withIncorrectNumberOfExitRequestsPerReport(uint256 maxRequestsCount) error when check is failed.  
function

checkExitBusOracleReport ( uint256 \_exitRequestsCount )

### Arguments

- \_exitRequestsCount
- — number of validator exit requests supplied per oracle report

### checkExitedValidatorsRatePerDay()

Validates that number of exited validators does not exceed the limit set byLimitsList.churnValidatorsPerDayLimit .

note Reverts withExitedValidatorsLimitExceeded(uint256 limitPerDay, uint256 exitedPerDay) error when check is failed.  
function

checkExitedValidatorsRatePerDay ( uint256 \_exitedValidatorsCount )

### Arguments

- \_exitedValidatorsCount
- — number of validator exit requests supplied per oracle report

### checkNodeOperatorsPerExtraDataItemCount()

Validates that number of node operators reported per extra data item does not exceed the limit set by `LimitsList.maxNodeOperatorsPerExtraDataItemCount` .

note Reverts with `TooManyNodeOpsPerExtraDataItem(uint256 itemIndex, uint256 nodeOpsCount)` error when check is failed. function

`checkNodeOperatorsPerExtraDataItemCount ( uint256 _itemIndex , uint256 _nodeOperatorsCount )`

### Arguments

- `_itemIndex`
- — index of item in extra data
- `_nodeOperatorsCount`
- — number of validator exit requests supplied per oracle report

### **checkAccountingExtraDataListItemsCount()**

Validates that number of extra data items in the report does not exceed the limit set by `LimitsList.maxAccountingExtraDataListItemsCount` .

note Reverts with `MaxAccountingExtraDataItemsCountExceeded(uint256 maxItemsCount, uint256 receivedItemsCount)` error when check is failed. function

`checkAccountingExtraDataListItemsCount ( uint256 _extraDataListItemsCount )`

### Arguments

- `_extraDataListItemsCount`
- — number of validator exit requests supplied per oracle report

### **checkWithdrawalQueueOracleReport()**

Validates that withdrawal request with the passed `_lastFinalizableRequestId` was created more than `LimitsList.requestTimestampMargin` seconds ago.

note Reverts with `IncorrectRequestFinalization(uint256 requestCreationBlock)` error when check is failed. function

`checkWithdrawalQueueOracleReport ( uint256 _lastFinalizableRequestId , uint256 _reportTimestamp )`

### Arguments

- `_lastFinalizableRequestId`
- — last finalizable withdrawal request id
- `_reportTimestamp`
- — timestamp when the originated oracle report was submitted

### **checkSimulatedShareRate()**

Applies sanity checks to the simulated share rate for withdrawal requests finalization.

note Reverts with `IncorrectSimulatedShareRate(uint256 simulatedShareRate, uint256 actualShareRate)` error when simulated share rate deviation exceeds the limit set by `LimitsList.simulatedShareRateDeviationBPLimit` function

`checkSimulatedShareRate ( uint256 _postTotalPooledEther , uint256 _postTotalShares , uint256 _etherLockedOnWithdrawalQueue , uint256 _sharesBurntDueToWithdrawals , uint256 _simulatedShareRate )`

### Arguments

- `_postTotalPooledEther`
- — total pooled ether after report applied
- `_postTotalShares`
- — total shares after report applied
- `_etherLockedOnWithdrawalQueue`
- — ether locked on withdrawal queue for the current oracle report
- `_sharesBurntDueToWithdrawals`
- — shares burnt due to withdrawals finalization
- `_simulatedShareRate`
- — share rate provided with the oracle report (simulated via off-chain "eth\_call")

## View Methods

### getLidoLocator()

Returns the address of the protocol-wide [LidoLocator](#) instance.

function

getLidoLocator ( )

returns

( address )

### getOracleReportLimits()

Returns the limits list used for the sanity checks as the [LimitsList](#) type.

function

getOracleReportLimits ( )

returns

( LimitsList memory )

### getMaxPositiveTokenRebase()

Returns max positive token rebase value with 1e9 precision (e.g.:1e6 — 0.1%;1e9 — 100%):

note Special values:

- 0
- (zero value) means uninitialized
- type(uint64).max
- means unlimited, e.g. not enforced Get max positive rebase allowed per single oracle report. Token rebase happens on total supply and/or total shares adjustment, while huge positive rebase can incur oracle report sandwiching stealing part of the stETH holders' rewards.

The relative positive rebase value derived as follows:

stETH balance for the account defined as:

balanceOf ( account )

= shares [ account ]

\* totalPooledEther / totalShares = shares [ account ]

\* shareRate Suppose shareRate changes when oracle reports (see handleOracleReport ) which means that token rebase happens:

## preShareRate

preTotalPooledEther ( )

/

preTotalShares ( ) postShareRate =

postTotalPooledEther ( )

/

postTotalShares ( ) R =

( postShareRate - preShareRate )

/ preShareRate here  $R > 0$  corresponds to the relative positive rebase value (i.e., instant APR).

function

getMaxPositiveTokenRebase ( )

returns

( uint256 )

## smoothenTokenRebase()

Evaluates the following amounts during Lido's oracle report processing:

- the allowed ETH amount that might be taken from the withdrawal vault and EL rewards vault
- the allowed amount of stETH shares to be burnt

function

smoothenTokenRebase ( uint256 \_preTotalPooledEther , uint256 \_preTotalShares , uint256 \_preCLBalance , uint256 \_postCLBalance , uint256 \_withdrawalVaultBalance , uint256 \_elRewardsVaultBalance , uint256 \_sharesRequestedToBurn , uint256 \_etherToLockForWithdrawals , uint256 \_newSharesToBurnForWithdrawals )

returns

( uint256 withdrawals , uint256 elRewards , uint256 simulatedSharesToBurn , uint256 sharesToBurn )

## Arguments

- \_preTotalPooledEther
- — total amount of ETH controlled by the protocol
- \_preTotalShares
- — total amount of minted stETH shares
- \_preCLBalance
- — sum of all Lido validators' balances on the Consensus Layer before the current oracle report
- \_postCLBalance
- — sum of all Lido validators' balances on the Consensus Layer after the current oracle report
- \_withdrawalVaultBalance
- — withdrawal vault balance on Execution Layer for the report calculation moment
- \_elRewardsVaultBalance
- — elRewards vault balance on Execution Layer for the report calculation moment
- \_sharesRequestedToBurn
- — shares requested to burn through Burner for the report calculation moment
- \_etherToLockForWithdrawals
- — ether to lock on withdrawals queue contract
- \_newSharesToBurnForWithdrawals
- — new shares to burn due to withdrawal requests finalization

## Returns

- withdrawals
- — ETH amount allowed to be taken from the withdrawals vault
- elRewards
- — ETH amount allowed to be taken from the EL rewards vault
- simulatedSharesToBurn
- — simulated amount of shares to be burnt (if no ether locked on withdrawals)
- sharesToBurn
- — amount of shares to be burnt (accounting for withdrawals finalization)

## Lever Methods

### setOracleReportLimits()

Sets the new values for the limits list.

note \* Requires `ALL_LIMITS_MANAGER_ROLE` \* to be granted to the caller. \* Reverts with `IncorrectLimitValue(uint256 value, uint256 minAllowedValue, uint256 maxAllowedValue)` \* error when some \* value in the passed data out of the allowed range. \* See details of allowed value boundaries in the [Limits List](#) \* section. function

setOracleReportLimits ( LimitsList memory \_limitsList )

## Arguments

- `_limitsList`
- — new limits list values

## setChurnValidatorsPerDayLimit()

Sets the new value for the `LimitsList.churnValidatorsPerDayLimit` . The limit is applicable for appeared and exited validators.

note \* Requires `CHURN_VALIDATORS_PER_DAY_LIMIT_MANAGER_ROLE` \* to be granted to the caller. \* Reverts with `IncorrectLimitValue()` \* error when the passed value is out of the allowed range. \* See [Limits List](#) \* section for details.  
function

`setChurnValidatorsPerDayLimit ( uint256 _churnValidatorsPerDayLimit )`

## Arguments

- `_churnValidatorsPerDayLimit`
- — `newLimitsList.churnValidatorsPerDayLimit`
- value

## setOneOffCLBalanceDecreaseBPLimit()

Sets the new value for the `LimitsList.oneOffCLBalanceDecreaseBPLimit` variable.

note \* Requires `ONE_OFF_CL_BALANCE_DECREASE_LIMIT_MANAGER_ROLE` \* to be granted to the caller. \* Reverts with `IncorrectLimitValue()` \* error when the passed value is out of the allowed range. \* See [Limits List](#) \* section for details.  
function

`setOneOffCLBalanceDecreaseBPLimit ( uint256 _oneOffCLBalanceDecreaseBPLimit )`

## Arguments

- `_oneOffCLBalanceDecreaseBPLimit`
- — new value for `LimitsList.oneOffCLBalanceDecreaseBPLimit`

## setAnnualBalanceIncreaseBPLimit()

Sets the new value for the `LimitsList.annualBalanceIncreaseBPLimit` variable.

note \* Requires `ANNUAL_BALANCE_INCREASE_LIMIT_MANAGER_ROLE` \* to be granted to the caller. \* Reverts with `IncorrectLimitValue()` \* error when the passed value is out of the allowed range. \* See [Limits List](#) \* section for details.  
function

`setAnnualBalanceIncreaseBPLimit ( uint256 _annualBalanceIncreaseBPLimit )`

## Arguments

- `_annualBalanceIncreaseBPLimit`
- — new value for `LimitsList.annualBalanceIncreaseBPLimit`

## setSimulatedShareRateDeviationBPLimit()

Sets the new value for the `LimitsList.simulatedShareRateDeviationBPLimit` variable.

note \* Requires `SHARE_RATE_DEVIATION_LIMIT_MANAGER_ROLE` \* to be granted to the caller. \* Reverts with `IncorrectLimitValue()` \* error when the passed value is out of the allowed range. \* See [Limits List](#) \* section for details.  
function

`setSimulatedShareRateDeviationBPLimit ( uint256 _simulatedShareRateDeviationBPLimit )`

## Arguments

- `_simulatedShareRateDeviationBPLimit`
- — new value for `LimitsList.simulatedShareRateDeviationBPLimit`

## setMaxExitRequestsPerOracleReport()

Sets the new value for theLimitsList.maxValidatorExitRequestsPerReport .

note \* RequiresMAX\_VALIDATOR\_EXIT\_REQUESTS\_PER\_REPORT\_ROLE \* to be granted to the caller. \* Reverts withIncorrectLimitValue() \* error when the passed value is out of the allowed range. \* See[Limits List](#) \* section for details. function

setMaxExitRequestsPerOracleReport ( uint256 \_maxValidatorExitRequestsPerReport )

### Arguments

- \_maxValidatorExitRequestsPerReport
- — new value forLimitsList.maxValidatorExitRequestsPerReport

### setRequestTimestampMargin()

Sets the new value for theLimitsList.requestTimestampMargin variable.

note \* RequiresREQUEST\_TIMESTAMP\_MARGIN\_MANAGER\_ROLE \* to be granted to the caller. \* Reverts withIncorrectLimitValue() \* error when the passed value is out of the allowed range. \* See[Limits List](#) \* section for details. function

setRequestTimestampMargin ( uint256 \_requestTimestampMargin )

### Arguments

- \_requestTimestampMargin
- — new new value forLimitsList.requestTimestampMargin

### setMaxPositiveTokenRebase()

Sets the new value for theLimitsList.maxPositiveTokenRebase variable.

note \* RequiresMAX\_POSITIVE\_TOKEN\_REBASE\_MANAGER\_ROLE \* to be granted to the caller. \* Reverts withIncorrectLimitValue() \* error when the passed value is out of the allowed range. \* See[Limits List](#) \* section for details. function

setMaxPositiveTokenRebase ( uint256 \_maxPositiveTokenRebase )

### Arguments

- \_maxPositiveTokenRebase
- — new value forLimitsList.maxPositiveTokenRebase

### setMaxAccountingExtraDataListItemsCount()

Sets the new value for theLimitsList.maxAccountingExtraDataListItemsCount variable.

note \* RequiresMAX\_ACCOUNTING\_EXTRA\_DATA\_LIST\_ITEMS\_COUNT\_ROLE \* to be granted to the caller. \* Reverts withIncorrectLimitValue() \* error when the passed value is out of the allowed range. \* See[Limits List](#) \* section for details. function

setMaxAccountingExtraDataListItemsCount ( uint256 \_maxAccountingExtraDataListItemsCount )

### Arguments

- \_maxAccountingExtraDataListItemsCount
- — new value forLimitsList.maxAccountingExtraDataListItemsCount

### setMaxNodeOperatorsPerExtraDataItemCount()

Sets the new value for theLimitsList.maxNodeOperatorsPerExtraDataItemCount variable.

note \* RequiresMAX\_NODE\_OPERATORS\_PER\_EXTRA\_DATA\_ITEM\_COUNT\_ROLE \* to be granted to the caller. \* Reverts withIncorrectLimitValue() \* error when the passed value is out of the allowed range. \* See[Limits List](#) \* section for details. function

setMaxNodeOperatorsPerExtraDataItemCount ( uint256 \_maxNodeOperatorsPerExtraDataItemCount )

### Arguments

- `_maxNodeOperatorsPerExtraDataItemCount`
- — new value for `LimitsList.maxNodeOperatorsPerExtraDataItemCount`

## Permissions

### ALL\_LIMITS\_MANAGER\_ROLE()

bytes32

public

constant ALL\_LIMITS\_MANAGER\_ROLE =

keccak256 ( "ALL\_LIMITS\_MANAGER\_ROLE" ) Granting this role allows updating ANY value of the Limits List. See [setOracleReportLimits\(\)](#) method.

Grant this role with caution and give preference to the granular roles described below.

### CHURN\_VALIDATORS\_PER\_DAY\_LIMIT\_MANAGER\_ROLE()

Granting this role allows updating the `churnValidatorsPerDayLimit` value of the [Limits List](#) . See the [setChurnValidatorsPerDayLimit\(\)](#) method.

bytes32

public

constant CHURN\_VALIDATORS\_PER\_DAY\_LIMIT\_MANAGER\_ROLE = keccak256 ( "CHURN\_VALIDATORS\_PER\_DAY\_LIMIT\_MANAGER\_ROLE" )

### ONE\_OFF\_CL\_BALANCE\_DECREASE\_LIMIT\_MANAGER\_ROLE()

Granting this role allows updating the `annualBalanceIncreaseBPLimit` value of the [Limits List](#) . See the [setOneOffCLBalanceDecreaseBPLimit\(\)](#) method.

bytes32

public

constant ONE\_OFF\_CL\_BALANCE\_DECREASE\_LIMIT\_MANAGER\_ROLE = keccak256 ( "ONE\_OFF\_CL\_BALANCE\_DECREASE\_LIMIT\_MANAGER\_ROLE" )

### ANNUAL\_BALANCE\_INCREASE\_LIMIT\_MANAGER\_ROLE()

Granting this role allows updating the `oneOffCLBalanceDecreaseBPLimit` value of the [Limits List](#) . See the [setAnnualBalanceIncreaseBPLimit\(\)](#) method.

bytes32

public

constant ANNUAL\_BALANCE\_INCREASE\_LIMIT\_MANAGER\_ROLE = keccak256 ( "ANNUAL\_BALANCE\_INCREASE\_LIMIT\_MANAGER\_ROLE" )

### SHARE\_RATE\_DEVIATION\_LIMIT\_MANAGER\_ROLE()

Granting this role allows updating the `simulatedShareRateDeviationBPLimit` value of the [Limits List](#) . See the [setSimulatedShareRateDeviationBPLimit\(\)](#) method.

bytes32

public

constant SHARE\_RATE\_DEVIATION\_LIMIT\_MANAGER\_ROLE = keccak256 ( "SHARE\_RATE\_DEVIATION\_LIMIT\_MANAGER\_ROLE" )

### MAX\_VALIDATOR\_EXIT\_REQUESTS\_PER\_REPORT\_ROLE()

Granting this role allows updating the `maxValidatorExitRequestsPerReport` value of the [Limits List](#) . See the [setMaxExitRequestsPerOracleReport\(\)](#) method.



bytes32

public

constant MAX\_VALIDATOR\_EXIT\_REQUESTS\_PER\_REPORT\_ROLE = keccak256 ( "MAX\_VALIDATOR\_EXIT\_REQUESTS\_PER\_REPORT\_ROLE" )

### **MAX\_ACCOUNTING\_EXTRA\_DATA\_LIST\_ITEMS\_COUNT\_ROLE()**

Granting this role allows updating the `maxAccountingExtraDataListItemCount` value of the [Limits List](#) . See the [setMaxAccountingExtraDataListItemCount\(\)](#) method.

bytes32

public

constant MAX\_ACCOUNTING\_EXTRA\_DATA\_LIST\_ITEMS\_COUNT\_ROLE = keccak256 ( "MAX\_ACCOUNTING\_EXTRA\_DATA\_LIST\_ITEMS\_COUNT\_ROLE" )

### **MAX\_NODE\_OPERATORS\_PER\_EXTRA\_DATA\_ITEM\_COUNT\_ROLE()**

Granting this role allows updating the `maxNodeOperatorsPerExtraDataItem` value of the [Limits List](#) . See the [setMaxNodeOperatorsPerExtraDataItemCount\(\)](#) method.

bytes32

public

constant MAX\_NODE\_OPERATORS\_PER\_EXTRA\_DATA\_ITEM\_COUNT\_ROLE = keccak256 ( "MAX\_NODE\_OPERATORS\_PER\_EXTRA\_DATA\_ITEM\_COUNT\_ROLE" )

### **REQUEST\_TIMESTAMP\_MARGIN\_MANAGER\_ROLE()**

Granting this role allows updating the `requestTimestampMargin` value of the [Limits List](#) . See the [setRequestTimestampMargin\(\)](#) method.

bytes32

public

constant REQUEST\_TIMESTAMP\_MARGIN\_MANAGER\_ROLE =  
keccak256 ( "REQUEST\_TIMESTAMP\_MARGIN\_MANAGER\_ROLE" )

### **MAX\_POSITIVE\_TOKEN\_REBASE\_MANAGER\_ROLE()**

Granting this role allows updating the `maxPositiveTokenRebase` value of the [Limits List](#) . See the [setMaxPositiveTokenRebase\(\)](#) method.

bytes32

public

constant MAX\_POSITIVE\_TOKEN\_REBASE\_MANAGER\_ROLE = keccak256 ( "MAX\_POSITIVE\_TOKEN\_REBASE\_MANAGER\_ROLE" )

## **Events**

### **ChurnValidatorsPerDayLimitSet()**

Emits whenever the value of the `LimitsList.churnValidatorsPerDayLimit` value is changed.

event

ChurnValidatorsPerDayLimitSet ( uint256 churnValidatorsPerDayLimit ) ;

### **Arguments**

- `churnValidatorsPerDayLimit`
- — new value of the `LimitsList.churnValidatorsPerDayLimit`

## OneOffCLBalanceDecreaseBPLimitSet()

Emits whenever the value of theLimitsList.oneOffCLBalanceDecreaseBPLimit value is changed.

event

OneOffCLBalanceDecreaseBPLimitSet ( uint256 oneOffCLBalanceDecreaseBPLimit ) ;

### Arguments

- oneOffCLBalanceDecreaseBPLimit
- — new value of theLimitsList.oneOffCLBalanceDecreaseBPLimit

## AnnualBalanceIncreaseBPLimitSet()

Emits whenever the value of theLimitsList.annualBalanceIncreaseBPLimit value is changed.

event

AnnualBalanceIncreaseBPLimitSet ( uint256 annualBalanceIncreaseBPLimit ) ;

### Arguments

- annualBalanceIncreaseBPLimit
- — new value of theLimitsList.annualBalanceIncreaseBPLimit

## SimulatedShareRateDeviationBPLimitSet()

Emits whenever the value of theLimitsList.simulatedShareRateDeviationBPLimit value is changed.

event

SimulatedShareRateDeviationBPLimitSet ( uint256 simulatedShareRateDeviationBPLimit ) ;

### Arguments

- annualBalanceIncreaseBPLimit
- — new value of theLimitsList.simulatedShareRateDeviationBPLimit

## MaxPositiveTokenRebaseSet()

Emits whenever the value of theLimitsList.maxPositiveTokenRebase value is changed.

event

MaxPositiveTokenRebaseSet ( uint256 maxPositiveTokenRebase ) ;

### Arguments

- annualBalanceIncreaseBPLimit
- — new value of theLimitsList.maxPositiveTokenRebase

## MaxValidatorExitRequestsPerReportSet()

Emits whenever the value of theLimitsList.maxValidatorExitRequestsPerReport value is changed.

event

MaxValidatorExitRequestsPerReportSet ( uint256 maxValidatorExitRequestsPerReport ) ;

### Arguments

- maxValidatorExitRequestsPerReport
- — new value of theLimitsList.maxValidatorExitRequestsPerReport

## MaxAccountingExtraDataListItemsCountSet()

Emits whenever the value of theLimitsList.maxAccountingExtraDataListItemsCount value is changed.

event

MaxAccountingExtraDataListItemCountSet ( uint256 maxAccountingExtraDataListItemCount ) ;

### Arguments

- maxAccountingExtraDataListItemCount
- — new value of theLimitsList.maxAccountingExtraDataListItemCount

### MaxNodeOperatorsPerExtraDataItemCountSet()

Emits whenever the value of theLimitsList.maxNodeOperatorsPerExtraDataItemCount value is changed.

event

MaxNodeOperatorsPerExtraDataItemCountSet ( uint256 maxNodeOperatorsPerExtraDataItemCount ) ;

### Arguments

- maxNodeOperatorsPerExtraDataItemCount
- — new value of theLimitsList.maxNodeOperatorsPerExtraDataItemCount

### RequestTimestampMarginSet()

Emits whenever the value of theLimitsList.requestTimestampMargin value is changed.

event

RequestTimestampMarginSet ( uint256 requestTimestampMargin ) ;

### Arguments

- requestTimestampMargin
- — new value of theLimitsList.requestTimestampMargin [Edit this page](#) [Previous](#) [LegacyOracle](#) [Next](#) [OracleDaemonConfig](#)