Smart Contracts Specification for Incentivized Locks (iLocks) - V1

Purpose

The purpose of this document is to clarify the core functionality of the iLocks by:

- Laying out the core data structures of the locks and rewards, explicating their purpose;
- Detailing the mechanisms which dictate the creation and interactions with the locks;

Overview

Actor	Actions	Contract
owner	1. setServiceFee: 2. whitelistOperator:	
operator	1. createLockSchedule: 2. withdraw: 3. setTreasury: 4. setOperator: 5. setBaseURI: 6. pauseLockSchedule: 7. resumeLockSchedule:	LockManager

lock user	1. lock: 2. release: 3. earlyRelease: 4. claim 5. safeTransferFrom:	LockToken

---

Smart Contract Architecture

This project uses a diamond upgradability pattern (EIP-2535)[https://eips.ethereum.org/EIPS/eip-2535] implmentation by solidstate[https://github.com/solidstate-network/solidstate-solidity/blob/master/contracts/proxy/diamond/Diamond.sol].

Data Types

LockSchedule

Defines how the user can lock their token and what are the incentives and costs associated.

Parameter	Type	Description
`initiated`	`bool`	Flags if LockSchedule exists
`paused`	`bool`	Flags if the LockSchedule is accepting deposits
`treasury`	`address`	Address to receive the fees
`tokenLocked`	`address`	The address of the token the user can lock up in the `LockSchedule`
`startingDate`	`uint256`	Timestamp which the `LockSchedule` start allowing the users to lock their tokens
`tokenIds`	`uint256[]`	Array of the ERC1155 tokens minted to represent the locked positions
`rewards`	`mapping(uint256 => Reward)`	Mapping of `Reward` structs that define the `rewardRate` the user will get when locking
`rewardsCount`	`uint256`	Number of `Rewards` in the `LockSchdule`
`fees`	`Fees`	`Fee` struct applicable to the `LockSchedule`

Reward

Defines how many tokens the users will receive when locking tokens in the `LockSchedule` according to their `lockDuration`.

Parameter	Type	Description
`rewardToken`	`address`	Reward token address
`revocable`	`bool`	Whether or not the `vestingSchedule` derived from this reward struct is revocable
`revoked`	`bool`	Flag if Reward is active in the `LockSchedule`
`vestingPeriodEqualToLockDuration`	`bool`	Whether or not the `vestingDuration` is equal to the duration of the lock
`variableRewards`	`mapping(uint256 => VariableRewards)`	Mapping to store how `rewardRate` will vary depending on the duration of the lock
`variableRewardsCount`	`uint256`	Number of `VariableRewards` in the `Reward`

VariableRewards

Parameter	Type	Description
`rewardDuration`	`uint256`	Time to reach `finalRate` in seconds
`vestingDuration`	`uint256`	Reward `vestingDuration` in case it is different from `lockDuration`.
`cliff`	`uint256`	Initial wait period before start increasing `rewardRate`, in seconds
`spike`	`uint256`	Amount to increase in the `rewardRate` at the beginning of `rewardDuration`
`finalRate`	`uint256`	Final `rewardRate` after `rewardDuration` has passed
`slicePeriodSeconds`	`uint256`	Duration of a slice variation of the `rewardRate` in seconds.

VestingSchedule

Defines how the user will receive his `rewardToken` over time.

Parameter	Type	Description
`initialized`	`bool`	Checks if vestingSchedule exists
`token`	`address`	Vesting schedule reward token
`beneficiary`	`address`	Defines who will received the rewards
`cliff`	`uint256`	Initial wait period before start vesting, in seconds
`start`	`uint256`	Start timestamp of the vesting period
`duration`	`uint256`	Duration of the vesting period in seconds
`slicePeriodSeconds`	`uint256`	Duration of a slice period for the `vestingSchedule` in seconds
`revocable`	`bool`	Whether or not the `vestingSchedule` is revocable
`amountTotal`	`uint256`	Total amount of tokens to be released at the end of the vesting
`released`	`uint256`	Amount of tokens released
`revoked`	`bool`	Whether or not the vesting has been revoked

Fees

Defines how lock users will be charged for locking, transferring and releasing tokens before `releaseDate`, along the duration of the lockSchedule.

Parameter	Type	Description
`serviceFee`	`uint256`	Fee paid to OptyFi when locking tokens
`originationFee`	`uint256`	Fee paid to `treasury` when locking tokens
`variableFees`	`mapping(uint256 => VariableFees)`	Fees that may vary depending on the time remaining to `releaseDate`
`variableFeesCount`	`uint256`	Number of `VariableFees` in `Fees`

VariableFees

Parameter	Type	Description

`remainingTime`	`uint256`	Time remaining to `releaseDate`
`earlyReleaseFee`	`uint256`	Percentage of the transfer value that goes to the `treasury`
`transferFee`	`uint256`	Percentage of the value released before the `releaseDate` that goes to the `treasury`

External and Public Functions

LockManager

Implements Lock Manager functionalities.

createLockSchedule	
The Lock Manager creates a new `LockSchedule`	
Input	• `tokenLocked` • `scheduleBaseURI` • `startingDate` • `releaseDates` • `Fees` • `Rewards[]`
Performs	• check for duplicate `releaseDates` • check if every `releaseDate` is after `startingDate` • check for duplicate `remainingTime` in the `Fees` parameter • check if fees (service, origination, earlyRelase and transfer) are less than `MAX_BPS` (100%) • increment `lockScheduleId` and sets the `LockSchedule` attributes • set the `releaseDates` in the `TokenLock` contract • set `lockSchedule`'s `baseURI` • set the `LockSchedule` `Fees` and `Rewards` struct

withdraw	
The `operator` withdraws reward tokens	
Input	• `token` • `beneficiary` • `amount`
Performs	• Check if `TokenVesting` contract has at least `amount`of `rewardToken` • transfer `amount` to `treasury`

pauseLockSchedule	
The `operator` pauses the `LockSchedule`, so it no longer allow locks	
Input	`lockScheduleId`
Performs	set `LockSchedule` `status` to `PAUSED`

resumeLockSchedule	
The `operator` activates a `PAUSED` `LockSchedule`	
Input	`lockScheduleId`
Performs	set `LockSchedule` `status` to `ACTIVE`

setTreasury	
The `operator` sets the `treasury` address	
Input	• `lockScheduleId` • `treasury`
Performs	set `LockSchedule` `treasury` to the given address

setOperator	
The `operator` can change the `operator` address	
Input	• `lockScheduleId` • `operator`
Performs	set `LockSchedule` `operator`to the given address

setBaseURI	
Sets lockSchedule's baseURI	
Input	• `lockScheduleId` • `newBaseURI`
Performs	• sets lockSchedule's baseURI

LockManager - View Functions

lockScheduleCount	
Return the number of lock schedules created	
Input	no parameter
Performs	return the latest value of `lockScheduleId` representing the number of lock schedules created

vestingScheduleTotalCount	
Return the total number of `VestingSchedules` created	
Input	no parameter
Performs	Return the lenght of `vestingScheduleId` array

lockSchedule	
Return lock schedule structure	
Input	`lockScheduleId`
Performs	Return `LockSchedule` struct of `lockScheduleId`

withdrawableRewards	
Return the amount of reward tokens withdrawable by the `operator`	
Input	`token`
Performs	Returns the amount of reward `token` not allocated in any `vestingSchedule` which can be withdrawn by the owner

LockToken

Implements Lock token functionalities

lock	
Users lock their tokens to receive the equivalent amount in locked token and have their `vestingSchedule` set	
Input	• `tokenId` • `amount`
Performs	• check if the there is a `initiated` `LockSchedule` for `tokenId` • check whether `releaseDate` has been reached • transfer `tokenLocked` to `TokenLock` contract • mint balance in the corresponding `tokenId` • set `vestingSchedule` according to `Rewards` of the `LockSchedule` and the `lockDuration`

release	
User release their tokens after `releaseDate` has passed	
Input	• `tokenId` • `amount`
Performs	• check whether the user has a balance equal or less than the `amount` to be released • check if `releaseDate` for the `tokenId` is in the past • burn the released amount of `tokenId` • transfer the underlying `tokenLocked` to `msg.sender` in the `TokenLock` contract

earlyRelease	
User release their tokens before `releaseDate` has passed paying the `earlyWithdrawFee`	
Input	• `tokenId` • `amount`
Performs	• check whether the user has a balance equal or less than the `amount` to be released • calculate the `earlyReleaseFee` according to the current `block.timestamp` • transfer the `earlyReleaseFee` to the `treasury` • burn the released amount of `tokenId` • transfer the remaining amount of `tokenLocked` to `msg.sender` in the `TokenLock`

claim	
User claims reward in `TokenVesting` contract	
Input	• `index` • `amount`
Performs	• check in the `TokenVesting` contract how many `VestingSchedules` the user has • calculate the releaseable amount • transfer the rewards to the user

safeTransferFrom	
Charge `transferFee` before performing the transfer	
Input	• `from` • `to` • `id` • `amount` • `data`

Performs	• get the `transferFee` • transfer the `transferFee` to `treasury` • transfer the remainder amount to `beneficiary`

LockToken - View Functions

balance	
Balance function of solidstate's ERC1155 contract	
Input	• `tokenId`
Performs	Return balance recorded in `balance[tokenId]` mapping on `ERC1155Storage` contract

uri	
Returns the URI for given `tokenId`	
Input	• `tokenId`
Performs	get the BaseURI of the `lockScheduleId` and concatenate with `tokenId`

remainingTimeLocked	
Returns the time remaining to `releaseDate` of `tokenId`	
Input	• `tokenId`
Performs	get `releaseDate` of `tokenId` and subtract `block.timestamp`

currentRewardRate	
Returns the `rewardRate` correspondent to the `lockDuration`	
Input	• `tokenId` • `rewardId`
Performs	get the `rewardId` `Reward` struct of the `LockShedule` of `tokenId` and calculates the `rewardRate` based on the `lockDuration`

currentFees	
Returns the `Fees` of `tokenId`	
Input	• `tokenId`
Performs	get `Fees` of the `LockSchedule` correspondent to `tokenId` at current `block.timestamp`

vestingScheduleCount	
Returns how many `VestingSchedules` an user has created	
Input	• `beneficiary`
Performs	Return the number of `VestingSchedules` created by `beneficiary`

getVestingSchedule
Returns the `VestingSchedule` struct for `beneficiary` at `index`

Input	• `index` • `beneficiary`
Performs	• get the `vestingScheduleId` for `index` and `beneficiary` • get `VestingSchedule` struct in `vestingSchedule` mapping of `TokenVestingStorage`

claimableRewards	
Returns the amount of reward tokens claimable by the user	
Input	`token`
Performs	Returns the amount of reward `token` not allocated in any `vestingSchedule` which can be withdrawn by the owner