Lockup Transfer Manager

- **Introduced in:** v3.0.0
- **Contract name:** Lockup Transfer Manager
- **Compatible ST Protocol version range:** 3.0.0 - *
- **Type:** Transfer Manager Module
- Associated LucidChart: https://www.lucidchart.com/documents/edit/6a32441e-b4ec-430b-8f8f-cc02bfaddc19?shared=true&

How it works

This module is used to limit the volume of tokens to be transacted by the affiliate/investors. That helps in avoiding the material impact on the SecurityToken prices. In other words, it assigns a lockup on the tokens of the affiliates/investor (mainly large amount of percentage of token holders) and only allows them to transact a certain amount of tokens within a given time period.

Key functionalities (as defined in the Smart Contract)

Initialization

This module is initialized with no parameters. That means during creation of this contract there's no need to call any type of `configure()` function.

Using Module

Step-1 : Add the LockupType first by using `addNewLockUpType()` with following parameters

_lockupAmount → Lockup amount = 100,000

_startTime → Start Time of the lockup = now

_lockUpPeriodSeconds . → Number of seconds lockup will remain active = 4 years(~ 4 * 365 * 24 * 3600).

_releaseFrequencySeconds → Number of seconds after which a tranche of tokens will be released (This is a recurring process) = 1 year (~ 1 * 365 * 24 * 3600)

_lockupName → Name of the lockup (Should be unique) = "a_lockup"

Step-2 : Assign “a_lockup” to Alice using the `addLockUpByName()` with following parameters

_userAddress → Address of employee/Affiliate whom lockup get assigned = 0xabc..

_lockupName → Name of the lockup = a_lockup

Day1: Alice tries to sell 100 tokens

Current Alice balance : 100,000

Calculate the totalRemainingLockedAmount for Alice : LockupAmount - unlockedAmount = 100,000 - 0 = 100,00

For transfer to proceed, currentBalanceOfAlice - _amount >= totalRemainingLockedAmount

100,000 - 100 >= 100,00 → false → transaction failed

Day 250: Alice tries to sell 10,000 tokens

Current Alice balance : 110,000 (Buy 10,000 tokens by any other means ex- Secondary market , OTC etc).

totalRemainingLockedAmount = 100,000 - 0 = 100,000

110,000 - 10,000 >= 100,000 —> True, Transaction processed

Day 731: Alice tries to sell 40,000 tokens

Current Alice balance : 100,000

totalRemainingLockedAmount = 100,000 - 50,000 (2 years passed) = 50,000

100,000 - 40,000 >= 50,000 —> True, Transaction processed

continues ....

If Alice has multiple lockups then `totalRemainingLockedAmount` will be calculated by looping all active lockups of Alice. The locked amounts for each lockup will be summed together.

Transfer Verification

- If `_from` address has been added to the lockups mapping (i.e restricted address in terms of token volume) then `executeTransfer` function performs a check to calculate the permitted or allowed volume of tokens to transact.
- Transaction is allowed only when the total remaining locked amount is less than equal to the amount remain after the transaction `((currentBalance.sub(_amount)) >= totalRemainingLockedAmount`) (#460 -

#462). While the total remaining locked amount is the aggregation of the remaining locked amounts per lockups for a given address (i.e `_from`).
- Lockup restrictions enter in effect as soon as a user is added to them. However, the vesting schedule (release of tokens) starts from `_startTime`.
- If investor/affiliate doesn't transact the total allowed volume of tokens of a particular period then the remaining amount of tokens will be added into the allowed quota of the following period.
- If `_from` address is the issuance address (i.e `address(0)`) then `executeTransfer` will ignored. That means the minting of tokens is not affected by the `LockupVolumeRestrictionTM`. (#89)

```
/**
@dev Used to verify the transfer transaction and prevent locke
d up tokens from being transferred
* @param _from Address of the sender
* @param _amount The amount of tokens to transfer
*/
function executeTransfer(address  _from, address /* _to*/, uin
t256  _amount, bytes calldata /* _data */) external returns(Re
sult) {
```

Add lockup type

```
/**
     * @notice Use to add the new lockup type
     * @param _lockupAmount Amount of tokens that need to loc
k.
     * @param _startTime When this lockup starts (seconds)
     * @param _lockUpPeriodSeconds Total period of lockup (sec
onds)
     * @param _releaseFrequencySeconds How often to release a
tranche of tokens (seconds)
     * @param _lockupName Name of the lockup
     */
    function addNewLockUpType(
        uint256 _lockupAmount,
        uint256 _startTime,
        uint256 _lockUpPeriodSeconds,
        uint256 _releaseFrequencySeconds,
        bytes32 _lockupName
    )
        external
        withPerm(ADMIN)
    {
```

NB - tokens are locked as soon as the Lockup is created, but only start vesting from _startTime.
NB - _releaseFrequencySeconds can be set to a number greater than _lockUpPeriodSeconds to disable vesting release cycles (all tokens will be released together when restriction ends).

For Batch transaction-

```
 /**
     * @notice Use to add the new lockup type
     * @param _lockupAmounts Array of amount of tokens that ne
ed to lock.
     * @param _startTimes Array of startTimes when this lockup
starts (seconds)
     * @param _lockUpPeriodsSeconds Array of total period of l
ockup (seconds)
     * @param _releaseFrequenciesSeconds Array of how often to
release a tranche of tokens (seconds)
     * @param _lockupNames Array of names of the lockup
     */
    function addNewLockUpTypeMulti(
        uint256[] _lockupAmounts,
        uint256[] _startTimes,
        uint256[] _lockUpPeriodsSeconds,
        uint256[] _releaseFrequenciesSeconds,
        bytes32[] _lockupNames
    )
        public
        withPerm(ADMIN)
    {
```

Assign the lockup to a investor/affiliate using the lockup type*

For a particular user-

```
/**
     * @notice Add the lockup to a user
     * @param _userAddress Address of the user
     * @param _lockupName Name of the lockup
     */
    function addLockUpByName(
        address _userAddress,
        bytes32 _lockupName
    )
        external
        withPerm(ADMIN)
    {
```

NB- It is not possible to add an investor to a already started lockup (i.e lockup startTime < now).

For batch transaction.

```
/**
```

```
     * @notice Add the lockup to a user
     * @param _userAddresses Address of the user
     * @param _lockupNames Name of the lockup
     */
    function addLockUpByNameMulti(
        address[] _userAddresses,
        bytes32[] _lockupNames
    )
        public
        withPerm(ADMIN)
    {
```

Note: `addLockUpByNameMulti()` will hit the block gas limit if the no. of addresses is more than 80 - 90 (It is an approximate figure not tested).

Create a new lockup for a investor/affiliate

```
/**
     * @notice Lets the admin create a volume restriction lock
up for a given address.
     * @param _userAddress Address of the user whose tokens sh
ould be locked up
     * @param _lockupAmount Amount of tokens that need to loc
k.
     * @param _startTime When this lockup starts (seconds)
     * @param _lockUpPeriodSeconds Total period of lockup (sec
onds)
     * @param _releaseFrequencySeconds How often to release a
tranche of tokens (seconds)
     * @param _lockupName Name of the lockup
     */
    function addNewLockUpToUser(
        address _userAddress,
        uint256 _lockupAmount,
        uint256 _startTime,
        uint256 _lockUpPeriodSeconds,
        uint256 _releaseFrequencySeconds,
        bytes32 _lockupName
    )
        external
        withPerm(ADMIN)
    {
```

It works similar as adding the lockupType first and assign the lockup type to a given address.

For batch transaction-

```
/**
     * @notice Lets the admin create multiple volume restriction lockups for multiple given addresses.
     * @param _userAddresses Array of address of the user whose tokens should be locked up
     * @param _lockupAmounts Array of the amounts that need to be locked for the different addresses.
     * @param _startTimes Array of When this lockup starts (seconds)
     * @param _lockUpPeriodsSeconds Array of total periods of lockup (seconds)
     * @param _releaseFrequenciesSeconds Array of how often to release a tranche of tokens (seconds)
     * @param _lockupNames Array of names of the lockup
     */
    function addNewLockUpToUserMulti(
        address[] _userAddresses,
        uint256[] _lockupAmounts,
        uint256[] _startTimes,
        uint256[] _lockUpPeriodsSeconds,
        uint256[] _releaseFrequenciesSeconds,
        bytes32[] _lockupNames
    )
        public
        withPerm(ADMIN)
    {
```

Remove the Lockup restriction of a particular investor/affiliate

An investor/affiliate can have more than one lockup restriction. So for removing a particular lockup restriction we need to pass the `lockupName` and the `address of the investor/affiliate` (whose lockup restriction needs to be removed).

```
 /**
     * @notice Lets the admin remove a user's lock up
     * @param _userAddress Address of the user whose tokens ar
e locked up
     * @param _lockupName Name of the lockup need to be remove
d.
     */
    function removeLockUpFromUser(address _userAddress, bytes3
2 _lockupName) external withPerm(ADMIN) {
```

For Batch functions

```
/**
     * @notice Use to remove the lockup for multiple users
     * @param _userAddresses Array of addresses of the user wh
ose tokens are locked up
     * @param _lockupNames Array of the names of the lockup th
at needs to be removed.
```

```
     */
    function removeLockUpFromUserMulti(address[] _userAddresse
s, bytes32[] _lockupNames) public withPerm(ADMIN){
```

Remove LockupType in a transaction

Lockup type can only be removed when a given type doesn't have any associated address with it.

```
/**
     * @notice Used to remove the lockup type
     * @param _lockupName Name of the lockup
     */
    function removeLockupType(bytes32 _lockupName) external wi
thPerm(ADMIN) {
```

For batch transaction -

```
 /**
     * @notice Used to remove the multiple lockup type
     * @param _lockupNames Array of the lockup names.
     */
    function removeLockupTypeMulti(bytes32[] _lockupNames) pub
lic withPerm(ADMIN) {
```

Note: `removeLockUpTypeMulti()` will hit the block gas limit if the no. of addresses is more than 75 (It is an approximate figure not tested).

Modify a lockup type

Like other functions, the issuer or the designated delegate having `ADMIN` permissions can modify the lockup restriction of a particular investor/affiliate with the help of -

```
/**
     * @notice Lets the admin modify a lockup.
     * @param _lockupAmount Amount of tokens that needs to be
locked
```

```
     * @param _startTime When this lockup starts (seconds)
     * @param _lockUpPeriodSeconds Total period of lockup (sec
onds)
     * @param _releaseFrequencySeconds How often to release a
tranche of tokens (seconds)
     * @param _lockupName name of the lockup that needs to be
modified.
     */
    function modifyLockUpType(
        uint256 _lockupAmount,
        uint256 _startTime,
        uint256 _lockUpPeriodSeconds,
        uint256 _releaseFrequencySeconds,
        bytes32 _lockupName
    )
        external
        withPerm(ADMIN)
    {
```

For a batch transaction -

```
/**
     * @notice Lets the admin modify a volume restriction lock
up for a multiple address.
     * @param _lockupAmounts Array of the amount of tokens tha
t needs to be locked for the respective addresses.
     * @param _startTimes Array of the start time of the locku
ps (seconds)
     * @param _lockUpPeriodsSeconds Array of unix timestamp fo
r the list of lockups (seconds).
     * @param _releaseFrequenciesSeconds How often to release
a tranche of tokens (seconds)
```

```
     * @param _lockupNames Array of the lockup names that need
s to be modified
     */
    function modifyLockUpTypeMulti(
        uint256[] _lockupAmounts,
        uint256[] _startTimes,
        uint256[] _lockUpPeriodsSeconds,
        uint256[] _releaseFrequenciesSeconds,
        bytes32[] _lockupNames
    )
        public
        withPerm(ADMIN)
    {
```

Note: `modifyLockUpTypeMulti()` will hit the block gas limit if the no. of addresses is more than 75 (It is an approximate figure not tested).

Getters

- `getLockUp(bytes32 _lockUpName)` use to get the details of a particular lockup for a given lockup name.
- `getLockedTokenToUser(address _userAddress)` use to get the total locked tokens for a given user. It will loop to all lockups for a given address and return sum of locked tokens per lockups.
- `getListOfAddresses(bytes32 _lockupName)` use to get the list of the users of a lockup type.
- `getAllLockups()` use to get the all lockups created by the issuer till now.
- `getLockupsNamesToUser(address _user)` use to get the list of the lockups for a given user.
- `getTokensByPartition(bytes32 _partition, address _tokenHolder, uint256 _additionalBalance)` use to get balance of the tokenHolder for a given partition.
- `getAllLockupData()` use to get the details of all the lockupTypes

Special considerations / notes

None

Troubleshooting / FAQs

None

Know Issues / bugs

None