Blacklist Transfer Manager (BTM)

- **Introduced in:** 3.0.0
- **Contract name:** BlacklistTransferManager
- **Type:** Transfer Manager Module
- **Compatible Protocol Version:** 3.0.0 - *
- **Associated LucidChart:** https://www.lucidchart.com/documents/edit/97d5e109-2948-4916-b86f-a32521e359bd/0?shared=true&

How it works

This module allows approved employees/issuers to create and establish an automated blacklist to prevent insider investors from selling thus de-risking price swings during time sensitive cases like earning calls. The blacklists can automatically become active/inactive at set recurring intervals.

Key functionalities (as defined in the Smart Contract)

Initialization

This module does not require any initialization.

Execute Transfer

Summary:

- If the module is paused or no blacklist is imposed on the sender, the transfer restrictions are skipped for that specific transfer.
- The module works by parsing through all the blacklists applied on the sender and returns the invalid transfer if any of the blacklist is currently active.
- Transfers are allowed to go through if none of the blacklists applied on the sender are active at that current moment.

```
/**
* @param _from Address of the sender
* @dev Restrict the blacklist address to transfer tokens
* if the current time is between the timeframe define for the
* blacklist type associated with the _from address
*/
    function executeTransfer(address _from, address /* _to */,
uint256 /* _amount */, bytes calldata /* _data */)
```

Managing blacklist types

Admins can add a new blacklist, modify an existing blacklist or delete a blacklist at any time using the below functions

Note:
`_startTime` is the timestamp of the beginning of the first instance of the blacklist.
`_endTime` is the timestamp of the end of the first instance of the blacklist.
`_repeatPeriodTime` is number of cooldown days after which the blacklist should be re activated. If the `_repeatPeriodTime == 0` makes the period non-repeating.
`_name` is the name of the blacklist type.

Add Blacklist Type

Summary: This function is used to add the specified blacklist type.

```
/**
    * @notice Used to add the blacklist type
    * @param _startTime Start date of the blacklist type
    * @param _endTime End date of the blacklist type
    * @param _blacklistName Name of the blacklist type
    * @param _repeatPeriodTime Repeat period of the blacklist
type in days
    */
    function addBlacklistType(uint256 _startTime, uint256 _end
Time, bytes32 _blacklistName, uint256 _repeatPeriodTime) publi
```

```
c withPerm(ADMIN)
```

Add Blacklist Type Multi

Summary: This function is used to add multiple blacklist types in one transaction while having the same permission as addBlackListType().

Requirements:

- _startTimes.length == _endTimes.length
- _endTimes.length == _blacklistNames.length
- _blacklistNames.length == _repeatPeriodTimes.length, "Input array's length mismatch"

```
/**
    * @notice Used to add the multiple blacklist type
    * @param _startTimes Start date of the blacklist type
    * @param _endTimes End date of the blacklist type
    * @param _blacklistNames Name of the blacklist type
    * @param _repeatPeriodTimes Repeat period of the blacklist
type
    */
    function addBlacklistTypeMulti(
        uint256[] memory _startTimes,
        uint256[] memory _endTimes,
        bytes32[] memory _blacklistNames,
        uint256[] memory _repeatPeriodTimes
    )
        public
        withPerm(ADMIN)
```

Modify Blacklist Type

Summary: This function is used to modify the details of a given blacklist type.

Requirements:

- blacklists[_blacklistName].endTime != 0, "Blacklist type doesn't exist"

```
/**
    * @notice Used to modify the details of a given blacklist
type
    * @param _startTime Start date of the blacklist type
    * @param _endTime End date of the blacklist type
    * @param _blacklistName Name of the blacklist type
    * @param _repeatPeriodTime Repeat period of the blacklist
type
    */
    function modifyBlacklistType(uint256 _startTime, uint256 _
endTime, bytes32 _blacklistName, uint256 _repeatPeriodTime) pu
blic withPerm(ADMIN)
```

Modify Blacklist Type Multi

Summary: This function is used to modify multiple existing blacklist types as well as allow it to have the same access permission as modifyBlacklistType().

Requirements:

- _startTimes.length == _endTimes.length
- _endTimes.length == _blacklistNames.length
- _blacklistNames.length == _repeatPeriodTimes.length, "Input array's length mismatch"

```
/**
    * @notice Used to modify the details of a given multpile b
lacklist types
    * @param _startTimes Start date of the blacklist type
    * @param _endTimes End date of the blacklist type
    * @param _blacklistNames Name of the blacklist type
    * @param _repeatPeriodTimes Repeat period of the blacklist
type
    */
    function modifyBlacklistTypeMulti(
        uint256[] memory _startTimes,
        uint256[] memory _endTimes,
        bytes32[] memory _blacklistNames,
        uint256[] memory _repeatPeriodTimes
    )
        public
        withPerm(ADMIN)
```

Delete Blacklist Type

Summary: This function allows the issuer to delete the existing blacklist type.

Requirements:

- blacklists[_blacklistName].endTime != 0, "Blacklist type doesn’t exist"
- blacklistToInvestor[_blacklistName].length == 0, "Investors are associated with the blacklist"

```
/**
    * @notice Used to delete the blacklist type
    * @param _blacklistName Name of the blacklist type
    */
    function deleteBlacklistType(bytes32 _blacklistName) publi
c withPerm(ADMIN)
```

Delete Blacklist Type Multi

Summary: This function is used to delete the multiple blacklist types in a single transaction.

```
/**
    * @notice Used to delete the multiple blacklist type
    * @param _blacklistNames Name of the blacklist type
    */
    function deleteBlacklistTypeMulti(bytes32[] memory _blackl
istNames) public withPerm(ADMIN)
```

Applying blacklist to investors

Allows admins to add or remove users from blacklists at any time using listed functions below:

Note:
`_investor` is the Ethereum address of the investor.
`_blacklistName` is the name of the blacklist.

Add Investor to Blacklist

Summary: This function is used to assign a specific blacklist type to an investor. The _blacklistName should be pre-existing in the contract storage, otherwise the transaction will fail.

Requirements:

- blacklists[_blacklistName].endTime != 0, "Blacklist type doesn't exist"
- investor != address(0), "Invalid investor address"
- investorToIndex[_investor][_blacklistName] == 0, "Blacklist already added to investor”

```
/**
    * @notice Used to assign the blacklist type to the investo
r
```

```
    * @param _investor Address of the investor
    * @param _blacklistName Name of the blacklist
    */
    function addInvestorToBlacklist(address _investor, bytes32
_blacklistName) public withPerm(ADMIN)
```

Add Investor to Blacklist Multi

Summary: This function is used to assign the blacklist type to multiple investors.

```
 /**
    * @notice Used to assign the blacklist type to the multipl
e investor
    * @param _investors Address of the investor
    * @param _blacklistName Name of the blacklist
    */
    function addInvestorToBlacklistMulti(address[] memory _inv
estors, bytes32 _blacklistName) public withPerm(ADMIN)
```

Add Multi Investor to Blacklist Multi

Summary: This function is used to assign the multiple blacklist type to the multiple investor.

Requirements:

- _investors.length == _blacklistNames.length, "Input array's length mismatch"

```
 /**
    * @notice Used to assign the multiple blacklist type to th
e multiple investor
    * @param _investors Address of the investor
    * @param _blacklistNames Name of the blacklist
    */
    function addMultiInvestorToBlacklistMulti(address[] memory
_investors, bytes32[] memory _blacklistNames) public withPerm
(ADMIN)
```

Add Investor To New Blacklist

Summary: This function is used to assign the new blacklist type to an investor. It works by creating a new blacklistType and assign it to the investor.

```
/**
    * @notice Used to assign the new blacklist type to the inv
estor
    * @param _startTime Start date of the blacklist type
    * @param _endTime End date of the blacklist type
    * @param _blacklistName Name of the blacklist type
    * @param _repeatPeriodTime Repeat period of the blacklist
type
    * @param _investor Address of the investor
    */
    function addInvestorToNewBlacklist(
        uint256 _startTime,
        uint256 _endTime,
        bytes32 _blacklistName,
        uint256 _repeatPeriodTime,
        address _investor
    ) public withPerm(ADMIN)
```

Delete Investor From All Blacklist(s)

Summary: This function is used to delete the investor from all of the associated blacklist types.

Requirements:

- investor != address(0), "Invalid investor address"

```
/**
    * @notice Used to delete the investor from all the associa
ted blacklist types
    * @param _investor Address of the investor
    */
    function deleteInvestorFromAllBlacklist(address _investor)
public withPerm(ADMIN)
```

Delete Investor From All Blacklist(s) Multi

Summary: This function is used to delete multiple investors from all the associated blacklist types.

```
/**
    * @notice Used to delete the multiple investor from all th
e associated blacklist types
    * @param _investor Address of the investor
    */
    function deleteInvestorFromAllBlacklistMulti(address[] mem
ory _investor) public withPerm(ADMIN)
```

Delete Investor From Blacklist

Summary: This function is used to delete the investor from the blacklist

Requirement:

- _investor != address(0), "Invalid investor address"
- _blacklistName != bytes32(0),"Invalid blacklist name”
- investorToBlacklist = blacklistName, "Investor not associated to the blacklist"

```
/**
```

```
    * @notice Used to delete the investor from the blacklist
    * @param _investor Address of the investor
    * @param _blacklistName Name of the blacklist
    */
    function deleteInvestorFromBlacklist(address _investor, by
tes32 _blacklistName) public withPerm(ADMIN)
```

Delete Multi Investors From Blacklist Multi

Summary: This function is used to delete multiple investors from a blacklist.

Requirements:

- _investors.length == _blacklistNames.length, "Input array's length mismatch"

```
/**
    * @notice Used to delete the multiple investor from the bl
acklist
    * @param _investors address of the investor
    * @param _blacklistNames name of the blacklist
    */
    function deleteMultiInvestorsFromBlacklistMulti(address[]
memory _investors, bytes32[] memory _blacklistNames) public wi
thPerm(ADMIN)
```

Get List Of Addresses

Summary: This function allows the issuer to get the list of the investors of a blacklist type.

Requirement:

- blacklists[_blacklistName].endTime != 0, "Blacklist type doesn't exist"

```
/**
    * @notice get the list of the investors of a blacklist type
    * @param _blacklistName Name of the blacklist type
    * @return address List of investors associated with the blacklist
    */
    function getListOfAddresses(bytes32 _blacklistName) external view returns(address[] memory)
```

Get Blacklist Names To User

Summary: This function allows the issuer/employee to get the list of the investors of a blacklist type.

```
/**
    * @notice get the list of the investors of a blacklist type
    * @param _user Address of the user
    * @return bytes32 List of blacklist names associated with the given address
    */
    function getBlacklistNamesToUser(address _user) external view returns(bytes32[] memory)
```

Get All Blacklists

Summary: This function allows the issuer to get a list of all the blacklist names.

```
/**
     * @notice get the list of blacklist names
     * @return bytes32 Array of blacklist names
     */
    function getAllBlacklists() external view returns(bytes32[] memory)
```

Summary: Use to get the balance of token holder for a given partition

```
/**
     * @notice return the amount of tokens for a given user as
per the partition
     * @param _partition Identifier
     * @param _tokenHolder Whom token amount need to query
     * @param _additionalBalance It is the `_value` that trans
fer during transfer/transferFrom function call
     */
    function getTokensByPartition(bytes32 _partition, address
_tokenHolder, uint256 _additionalBalance) external view return
s(uint256)
```

Special considerations / notes

- function `addInvestorToBlacklistMulti` is limited by GAS limit and can only process a maximum number of address at a time.

- To set a non recurring blacklist, `_repeatPeriodTime` can be set to a zero.

- All investors must be removed from a blacklist before that blacklist can be deleted using `deleteBlacklistType` function.

Troubleshooting / FAQs

None

Know Issues / bugs

None

Changelog

Fixed:

- Blacklist periods now begin x days after start time (and not end time as before). This allows the repeat period to start from the start time rather than the end time. (Note: Having 0 repeat periods is allowed)
- `addInvestorToBlacklist` now allows you to only add an investor once