General Transfer Manager

- **Contract name:** GeneralTransferManager.sol
- **Type:** Transfer Manager Module
- **Compatible Protocol Version:** TBD
- **Associated LucidChart**: TBD

How it works

Summary: The General Transfer Manager module is for core transfer validation functionality. This module manages the transfer restrictions for the Security Token. By default, the GMT allows for only whitelisted addresses will be able to buy or sell the tokens.

GeneralTransferManager will check 5 things before approving a transfer:

1. That both the sender and receiver are in its internal whitelist.
2. That the seller is not subject to sale restrictions imposed by securities laws.
3. That the buyer is not subject to purchase restrictions imposed by securities laws.
4. That both the buyer and seller's KYC verification hasn't expired.
5. That, if it is a transfer from STO operation, that the buyer has been permitted to buy from it.

Key functionalities (as defined in the Smart Contract)

get Init Function

Summary: This function returns the signature of the configure function.

```
/**
 * @dev returns the signature configure function
 * @param
 */
```

```
 function getInitFunction() public pure returns (bytes4) {
    return bytes4(0);
}
```

changeIssuanceAddress

Summary: This function is used to change the issuance addresses of your STO.

```
/**
* @dev Used to change the Issuance Address
* @param _issuanceAddress new address for the issuance
*/

function changeIssuanceAddress(address _issuanceAddress) publi
c withPerm(FLAGS) {
issuanceAddress = _issuanceAddress;
emit ChangeIssuanceAddress(_issuanceAddress);

}
```

changeSigningAddress

Summary: This function is used to change the signing address and/or creating a new addresses for the signing

```
/**

* @dev change the Sigining Address
* @param _signingAddress new address for the signing

*/
```

```
function changeSigningAddress(address _signingAddress) public
withPerm(FLAGS) {
  signingAddress = _signingAddress;
  emit ChangeSigningAddress(_signingAddress);
}
```

changeAllowAllTransfers

Summary: This function is used to change the flag for allowing or not allowing transfers

```
/**
* @dev use to change the flag
true - It refers there are no transfer restrictions, for any a
ddresses
false - It refers transfers are restricted for all addresses.
* @param _allowAllTransfers flag value
*/
function changeAllowAllTransfers(bool _allowAllTransfers) publ
ic withPerm(FLAGS) {
  allowAllTransfers = _allowAllTransfers;
  emit AllowAllTransfers(_allowAllTransfers);
}
```

changeAllowAllWhitelistTransfers

Summary: This function is used to change the flag (true or false). As you can see below the true flag indicates that the time lock is ignored for transfers and that the address must be on the whitelist. The false flag means that transfers are restricted for all addresses.

```
/**
* @dev Used to change the flag
```

```
    true - It refers that time lock is ignored for transfers
(address must still be on whitelist)
    false - It refers transfers are restricted for all address
es.
* @param _allowAllWhitelistTransfers flag value
*/
    function changeAllowAllWhitelistTransfers(bool _allowAllWh
itelistTransfers) public withPerm(FLAGS) {
        allowAllWhitelistTransfers = _allowAllWhitelistTransfe
rs;
        emit AllowAllWhitelistTransfers(_allowAllWhitelistTran
sfers);
    }
```

changeAllowAllWhitelistIssuances

Summary: This function is used to change the flag (true or false). As you can see below the true flag indicates that the time lock is ignored for issuances and that the address must be on the whitelist. The false flag means that transfers are restricted for all addresses.

```
/*
* @notice Used to change the flag
    true - It refers that time lock is ignored for issuances
(address must still be on whitelist)
    false - It refers transfers are restricted for all address
es.
* @param _allowAllWhitelistIssuances flag value
*/

   function changeAllowAllWhitelistIssuances(bool _allowAllWhi
telistIssuances) public withPerm(FLAGS) {
```

```
        allowAllWhitelistIssuances = _allowAllWhitelistIssuance
s;
        emit AllowAllWhitelistIssuances(_allowAllWhitelistIssua
nces);

    }
```

changeAllowAllBurnTransfers

Summary: This function is used to change the flag (true or false). As you can see below the true flag indicates that tokens can be burned. The false flag triggers the burning mechanism to be deactivated.

```
/**
* @dev Used to change the flag
    true - It allow to burn the tokens
    false - It deactivate the burning mechanism.
* @param _allowAllBurnTransfers flag value
*/
    function changeAllowAllBurnTransfers(bool _allowAllBurnTra
nsfers) public withPerm(FLAGS) {
        allowAllBurnTransfers = _allowAllBurnTransfers;
        emit AllowAllBurnTransfers(_allowAllBurnTransfers);
    }
```

verifyTransfer

Summary: The `verifyTransfer` function uses the `GeneralTransferManager`'s whitelist to determine if the transfer between these two accounts can happen. Essentially this function is called when there is a transfer request from the STO. When called, it checks to see that the investor is in the whitelist.

`verifyTransfer()` is called to check whether the _from & _to addresses are in the whitelist or not.

1. In the case of an STO, minted tokens can only be transferred to the to investor when `allowAllWhitelistIssuances` flag is set to true and `_from` address should be issuanceAddress (~0x0).
2. In order for a transfer to occur between 2 whitelisted investors, two conditions need to pass before the transfer can be successful:
   a. `allowAllWhitelistTransfers` flag should be set to true.
   b. Anyone on the whitelist can transfer provided the blockNumber is large enough. In order for this to occur the seller's(_from) sell lockup period needs to be over and the buyer's (_to) purchase lockup period needs to be over.
3. If issuer set `allowAllTransfer` flag is set to true then all transfers are allowed, regardless of whitelist. However, this is not a method we recommend when handling the transfer of securities.
4. When the _to address is 0x0 (the Burn token call) there needs to be one condition satisfied:
   a. `allowAllBurnTransfers` should be set to true, otherwise the Burn call will not be executed.

Note: When the contract is paused, it will not check any of the above conditions.

```
/**
* @dev Used to verify the transfer transaction and prevent loc
ked up tokens from being transferred
* If the transfer request comes from the STO, it only checks t
hat the investor is in the whitelist
* If the transfer request comes from a token holder, it checks
that:
* a) Both are on the whitelist
* b) Seller's sale lockup period is over
* c) Buyer's purchase lockup is over

* @param _from Address of the sender
```

```
* @param _to Address of the receiver
*/

    function verifyTransfer(address _from, address _to, uint256 /*_amount*/, bytes /* _data */, bool /* _isTransfer */) public returns(Result) {
        if (!paused) {
            if (allowAllTransfers) {
                //All transfers allowed, regardless of whitelist
                return Result.VALID;
            }
            if (allowAllBurnTransfers && (_to == address(0))) {
                return Result.VALID;
            }
            if (allowAllWhitelistTransfers) {
                //Anyone on the whitelist can transfer, regardless of time
                return (_onWhitelist(_to) && _onWhitelist(_from)) ? Result.VALID : Result.NA;
            }
            if (allowAllWhitelistIssuances && _from == issuanceAddress) {
                if (!whitelist[_to].canBuyFromSTO && _isSTOAttached()) {
                    return Result.NA;
                }
                return _onWhitelist(_to) ? Result.VALID : Result.NA;
            }
```

modifyWhitelist

Summary: This function is used to add or remove addresses from a whitelist.

```
/**
* @dev Adds or removes addresses from the whitelist.
* @param _investor is the address to whitelist
* @param _fromTime is the moment when the sale lockup period e
nds and the investor can freely sell his tokens
* @param _toTime is the moment when the purchase lockup period
ends and the investor can freely purchase tokens from others
* @param _expiryTime is the moment till investors KYC will be
validated. After that investor need to do re-KYC
* @param _canBuyFromSTO is used to know whether the investor i
s restricted investor or not.
    */
    function modifyWhitelist(
        address _investor,
        uint256 _fromTime,
        uint256 _toTime,
        uint256 _expiryTime,
        bool _canBuyFromSTO
    )


//Passing a _time == 0 into this function, is equivalent to re
moving the _investor from the whitelist
```

modifyWhitelistMulti

Summary: This function is used to add or remove multiple addresses from the whitelist.

```
/**
* @dev Adds or removes addresses from the whitelist.
* @param _investors List of the addresses to whitelist
* @param _fromTimes An array of the moment when the sale locku
p period ends and the investor can freely sell his tokens
* @param _toTimes An array of the moment when the purchase loc
kup period ends and the investor can freely purchase tokens fr
om others
* @param _expiryTimes An array of the moment till investors KY
C will be validated. After that investor need to do re-KYC
* @param _canBuyFromSTO An array of boolean values
*/
    function modifyWhitelistMulti(
        address[] _investors,
        uint256[] _fromTimes,
        uint256[] _toTimes,
        uint256[] _expiryTimes,
        bool[] _canBuyFromSTO
    )
```

modifyWhitelistSigned

Summary: This function can be called by anyone who has a valid signature and allows that person to add or remove multiple addresses from the whitelist.

```
/**
* @dev Adds or removes addresses from the whitelist - can be c
alled by anyone with a valid signature
* @param _investor is the address to whitelist
```

```
* @param _fromTime is the moment when the sale lockup period e
nds and the investor can freely sell his tokens
* @param _toTime is the moment when the purchase lockup period
ends and the investor can freely purchase tokens from others
* @param _expiryTime is the moment till investors KYC will be
validated. After that investor need to do re-KYC
* @param _canBuyFromSTO is used to know whether the investor i
s restricted investor or not.
* @param _validFrom is the time that this signature is valid f
rom
* @param _validTo is the time that this signature is valid unt
il
* @param _nonce nonce of signature (avoid replay attack)
* @param _v issuer signature
* @param _r issuer signature
* @param _s issuer signature

*/
    function modifyWhitelistSigned(
        address _investor,
        uint256 _fromTime,
        uint256 _toTime,
        uint256 _expiryTime,
        bool _canBuyFromSTO,
        uint256 _validFrom,
        uint256 _validTo,
        uint256 _nonce,
        uint8 _v,
        bytes32 _r,
        bytes32 _s
    )
```

_checkSig

Summary: This function is simply used to verify the signature.

```
/**
 * @dev verify the signature
*/
    function _checkSig(bytes32 _hash, uint8 _v, bytes32 _r, by
tes32 _s) internal view {

//Check that the signature is valid
//sig should be signing - _investor, _fromTime, _toTime & _exp
iryTime and be signed by the issuer address
```

_onWhitelist

Summary: This function is an internal function that is used to check whether the investor is in the whitelist or not as well as check whether the KYC of investor get expired or not.

```
/**
* @dev check whether the investor is in the whitelist or not &
also checks whether the KYC of investor get expired or not
* @param _investor Address of the investor

*/
    function _onWhitelist(address _investor) internal view ret
urns(bool) {
```

```
          return (((whitelist[_investor].fromTime != 0) || (wh
itelist[_investor].toTime != 0)) && (whitelist[_investor].expi
ryTime >= now)); /*solium-disable-line security/no-block-membe
rs*/


    }
```

_isSTOAttached

Summary: This function is an internal function that is used to tell us whether or not the STO is attached to the security token or not.

```
/**
* @notice use to know whether the STO is attached or not
*/
    function _isSTOAttached() internal view returns(bool) {
        bool attached = ISecurityToken(securityToken).getModul
esByType(3).length > 0;
        return attached;
    }
```

getPermissions

Summary: This function is used to tell us the permissions flags that are associated with the `GeneralTransferManager`.

```
/**
* @notice Return the permissions flag that are associated with
general transfer manager
*/
```

```
    function getPermissions() public view returns(bytes32[]) {
        bytes32[] memory allPermissions = new bytes32[](2);
        allPermissions[0] = WHITELIST;
        allPermissions[1] = FLAGS;
        return allPermissions;
    }
```

Special considerations / notes

None

Troubleshooting / FAQs

None

Know Issues / bugs

None

Changelog

(Pre-v1.4.0)

Added:

- `LogModifyWhitelist` event of GeneralTransferManager emit two more variables. i.e address which added the investor in whitelist(`_addedBy`) and records the timestamp at which modification in whitelist happen(`_dateAdded`).
- `permissions()` function added in GeneralTransferManager to get all permissions.

Changed/Fixed:

- `GeneralTransferManager` takes only 1 variables as constructor argument i.e address of the `securityToken`.
- Functions of `GeneralTransferManager` earlier controlled by the owner only, now those can be controlled by the delegates as well with having proper permissions.

Removed:

- Removed the `Delegable.sol, AclHelpers.sol, DelegablePorting.sol` contracts. Now permission manager factory takes their place . * `delegates` mapping removed from the `GeneralTransferManager`.

2.1.0

- **Changed** the data types of the whitelist to reduce the gas needed to store an entry -> Introduces breaking change since the data types will not match
- **Changed:** Now there is a proxy pattern to reduce deployment costs, bringing it down by 2mm gas approx.
- **Changed:** the whitelist data is now exposed so it can be read directly from the contract instead of relying on events.
- **Added** `address[] public investors` to record a list of all addresses that have been added to the whitelist
- **Fixed** for when `allowAllWhitelistIssuances` is FALSE
- **Added** clearer logic around toTime, fromTime & expiryTime