ERC721 Delegate Extension Technical Documentation

Smart Contract(s): ERC721Delegate.sol

Overview & Purpose:

The ERC721Delegate.sol contract is an extension of the standard OpenZeppelin ERC721Enumerable.sol extension, which is an extension of the standard ERC721 NFT. ERC721Delegate leverages the same logic of the ERC721Enumerable, to allow owners of an NFT to Delegate a specific token to another address. Delegating a token to another address does not physically move the token, it simply records a state and storage change that the token has been delegated to another address.

The purpose of delegating a token to another address is for voting and governance. We have seen a rise in token voting, both on-chain and off-chain, and using NFTs as a new mechanism for voting. The ERC721Delegate allows an owner of an NFT to maintain ownership, but effectively delegate their token to another address, such that if the governance contract or snapshot setup allows delegated NFT voting rights, the delegatee of that token can now vote on behalf of the token owner.

For Hedgey, this ERC721Delegate contract is used as the primary backbone for its StreamingNFT and StreamVestingNFT contracts. They leverage the power of the ERC721Delegate contract so that the contracts allow users to delegate their NFTs to another wallet, whereupon a smart contract or snapshot strategy can be used to view the delegated balances of the tokens locked in each vesting NFT, and delegates can vote with those ERC20 balances locked inside the NFTs, which they have been delegated.

Smart Contract Architecture & Functions

The ERC721Delegate.sol contract is not meant to be deployed by itself, but rather should be inherited by a final contract that instantiates it to leverage the power of the Enumerable and Delegation functions, to make a new NFT Collection.

The ERC721Delegate contract works very similarly to the ERC721Enumerable contract in that it uses a beforeTokenTransfer hook to add or remove tokens to the delegation enumerations. The logic is very similar to that of the enumeration logic, where when a token is minted or transferred to a new address, the token is delegated to that address and added to an index that can be iterated over. When a token is burned or transferred, the current delegate is removed via token being removed from the delegates index. It is important to change the delegate during a token transfer, because should the NFT be sold or transferred to an unrelated party, the new owner should not still have their token delegated to the previous owners delegate, or unwanted voting may occur.

Functions:

Primary Internal function for delegation, used by inherited contract:

```
    function _delegateToken(address delegate, uint256 tokenId) internal
```

The _delegateToken function is an internal function that should be called by the final external NFT contract, using this internal function to delegate each token to a new delegate

address. The function checks that only the owner of the token can call this function. An address that is a delegate of the token may not call this function, only the owner can assign delegates for its token.

Public Read functions:

```
function balanceOfDelegate(address delegate) public view returns (uint256)
```

The balanceOfDelegate function is a simple function that returns the count of the tokens that have been delegated to a specific address. Each time a new token is delegated to the address, the balance is incremented by 1, and each time a the token is removed from that delegate, its balance is subtracted by 1.

```
function delegatedTo(uint256 tokenId) public view returns (address)
```

delegatedTo function is a public function to view who the current delegate is of a specific tokenId. If the tokenId has not been assigned using the _delegateToken function, then the tokenId is delegated to the owner of the token.

```
function tokenOfDelegateByIndex(address delegate, uint256 index) public view returns (uint256)
```

tokenDelegateByIndex function is a powerful function that assists in the enumeration and iteration of tokens for each delegate. Within the storage, each token is mapped to a specific index, so that when an external contract wants to evaluate each tokenId that has been delegated to a specific address, the contract can leverage this function to iterate through the balanceOf the delegate, and receive the tokenId at each index (starting from 0 to < balanceOf).

Private Functions called in the beforeTokenTransfer Hook

```
function _addDelegate(address to, uint256 tokenId) private {
```

The _addDelegate function is a strictly private function, used by the contract during the beforeTokenTransfer hook to add a token to the new delegates enumeration. This function is called when a token is minted, or separately by the _transferDelegate private function when a token is transferred from one address to another address (where neither is the 0x0 address). The function will assign the tokenId to the end of the delegates index, delegate the token to the to address, and update their delegatedBalances.

```
function _removeDelegate(uint256 tokenId) private
```

The _removeDelegate function is also called in the beforeTokenTransfer hook to remove a tokenId from the current delegate's enumeration. It is called by the hook when a token is burned, or by the _transferDelegate() private function when a token is transferred from one address to another address (where neither is the 0x0 address). This function reduces the balance of the delegate, and removes the tokenId from the mapping and enumeration.

```
function _transferDelegate(address to, uint256 tokenId) private
```

The _transferDelegate function first calls the _removeDelegate function to remove the tokenId from the prior delegates mapping, and then adds the tokenId to the new delegates enumeration with the _addDelegate function. The _transferDelegate function is called by the beforeTokenTransfer hook when a token is transferred from one address to another, where neither address is the 0x0 address (ie not minted and not burned).