Project Hydro (HydroBlockchain)

About

Hydro is the fintech blockchain providing a security and identity protocol for 2FA, MFA, KYC, payments, and documents.
Project Hydro is **web3**, multi-chain decentralized ecosystem which uses cutting-edge cryptographic technology to secure user identities, accounts, and transactions. Hydro aims to **decentralize** financial services by bringing public blockchain technology to traditional private systems. It also allows developers around the world to boost their platforms and applications with this innovative technology.

Hydro is an open source blockchain project comprising of multiple smart contracts (protocols) and a dApp store, all powered by the HYDRO token.

IceFMS.sol
It is a library built to creating sorting order for maximizing space utilization for robust File Management System.

The contract was written in version 0.5.1. The pragma keyword was used to enable compiler checks.

Five files were imported which includes:
1. SafeMath.sol
Mathematical operations with safety checks that revert on errors like overflow for multiplication, division by zero, overflow for addition and subtraction and dividing by zero for modulus.

2. SafeMath8.sol
Same as SafeMath.sol but for uint8.

3. IceGlobal.sol
handle critical file management function. Function that helps to get global items info, to get global items stamping info from the entire File Management System of Ice. It also contains file to get global iems of a file, global indexes of a files, reserve and return global item slot, add items to the global items, delete items, transfer history of EINs and many more.

This is where all functions related to the global scope of ice is stored.

IceGlobal.sol imported **IdentityRegistryInterface.sol**: this is an interface name IdentityRegistryInterface. It contains isSigned function, Identity View Functions, Identity Management Functions, Recovery Management Functions.

4. IceSort.sol
this is the core file that handles the sorting of files using LinkedList. It is used to preserve the order of the files and check the validity of the order for the arrangement of the files

5. IceFMSAdv.sol

this imported SafeMath.sol, IceGlobal.sol and IceSort.sol

```
15      using SafeMath for uint;
16
17      using IceGlobal for IceGlobal.GlobalRecord;
18      using IceGlobal for IceGlobal.Association;
19      using IceGlobal for IceGlobal.UserMeta;
20      using IceGlobal for mapping (uint => bool);
21      using IceGlobal for mapping (uint8 => IceGlobal.ItemOwner);
22      using IceGlobal for uint;
23
24      using IceSort for mapping (uint => IceSort.SortOrder);
```

This is using all the libraries imported into he contract.

Events:

```
29      // When Sharing is completed
30      event SharingCompleted(uint EIN, uint index1, uint index2, uint recipientEIN);
31
32      // When Sharing is rejected
33      event SharingRejected(uint EIN, uint index1, uint index2, uint recipientEIN);
34
35      // When Sharing is removed
36      event SharingRemoved(uint EIN, uint index1, uint index2, uint recipientEIN);
```

i. event SharingCompleted(uint EIN, uint index1, uint index2, uint recipientEIN): is emitted when the sharing is completed.
ii. event SharingRejected(uint EIN, uint index1, uint index2, uint recipientEIN); is emitted when the sharing is rejected.
iii. event SharingRemoved(uint EIN, uint index1, uint index2, uint recipientEIN); is emitted when the sharing is removed.

function shareItemToEINs()

This function is used to share an item to many users, and this function is usually called by the owner of the item to be shared. It has eight parameters which includes:

i. **self**: is the mappings of all pointer to the GlobalRecord Struct (IceGlobal Library) which forms shares in Ice Contract. The GlobalRecord struct has member uint i1 and uint i2, used to access global index 1 and 2.

ii. **_globalItems**: this is the mappings of all items stored by all users.

iii. **_totalShareOrderMapping**: mapping of the entire shares order using SortOrder Struct (IceSort Library).

iv. **_shareCountMapping**: mapping of all share count.

v. **_blacklist**: is the entire mapping of the Blacklist for all users.

vi. **_rec**: is the GlobalRecord Struct (IceGlobal Library).

vii. **_ein** is the primary user who initiates sharing of the item.

` viii. **_toEINs** is the array of the users with whom the item will be shared.

function _shareItemToEIN()

This function is used to share an item to a specific user, and this function is usually called by the owner of the item to be shared. It has five parameters which includes:

i. **self**: is the mappings of all shares associated with the recipient user.
ii. **_globalItems**: is the mapping of all items stored by all users in the Ice FMS
iii. **_shareOrder**: is the mapping of the shares order using SortOrder Struct (IceSort Library) of the recipient user
iv. **_shareCount**: is the mapping of all share count
v. **_rec**: is the GlobalRecord Struct (IceGlobal Library)
vi. **_toEIN**: is the ein of the recipient user

And there are many more functions in the library.

IceFMS Library

```
18      using SafeMath for uint;
19      using SafeMath8 for uint8;
20
21      using IceGlobal for IceGlobal.GlobalRecord;
22      using IceGlobal for IceGlobal.Association;
23      using IceGlobal for IceGlobal.UserMeta;
24      using IceGlobal for mapping (uint => bool);
25      using IceGlobal for mapping (uint8 => IceGlobal.ItemOwner);
26      using IceGlobal for mapping (uint => mapping (uint => IceGlobal.Association));
27      using IceGlobal for uint;
28
29      using IceSort for mapping (uint => IceSort.SortOrder);
30      using IceSort for IceSort.SortOrder;
31
32      using IceFMSAdv for mapping (uint => mapping(uint => IceGlobal.GlobalRecord));
```

This is using all the libraries imported into he contract.

Structs

```
38      /* To define the multihash function for storing of hash */
39      struct FileMeta {
40          bytes32 name; // to store the name of the file
41
42          bytes32 hash; // to store the hash of file
43          bytes22 hashExtraInfo; // to store any extra info if required
44
45          bool encrypted; // whether the file is encrypted
46          bool markedForTransfer; // Mark the file as transferred
47
48          uint8 protocol; // store protocol of the file stored | 0 is URL, 1 is IPFS
49          uint8 transferCount; // To maintain the transfer count for mapping
50
51          uint8 hashFunction; // Store the hash of the file for verification | 0x000 for deleted files
52          uint8 hashSize; // Store the length of the digest
53
54          uint32 timestamp;  // to store the timestamp of the block when file is created
55      }
```

This is a struct called “FileMeta” which define the multihash function for storing of hash. The members of the struct includes:

i. bytes32 name: stores the name of the file
ii. bytes32 hash: stores the hash of the file
iii. bytes22 hashExtraInfo: stores any extra info for the hash of the file
iv. bool encrypted: whether the file is encrypted or not
v. bool markedForTransfer: mark the file as transferred. The transfer status of the file.
vi. uint8 protocol: stores the protocol of the file stored, where; 0 is URL, 1 is IPFS.
vii. uint8 transferCount: to maintain the transfer count for mapping
viii. uint8 hashFunction: store the hash of the file for verification, where 0x000 is for deleted files.
ix. uint8 hashSize: store the length of the digest; i.e the length of the hash
x. uint32 timestamp: to store the timestamp of the block when file is created

File

```
57      /* To define File structure of all stored files */
58      struct File {
59          // File Meta Data
60          IceGlobal.GlobalRecord rec; // store the association in global record
61
62          // File Properties
63          bytes protocolMeta; // store metadata of the protocol
64          FileMeta fileMeta; // store metadata associated with file
65
66          // File Properties - Encryption Properties
67          mapping (address => bytes32) encryptedHash; // Maps Individual address to the stored hash
68
69          // File Other Properties
70          uint associatedGroupIndex; // to store the group index of the group that holds the file
71          uint associatedGroupFileIndex; // to store the mapping of file in the specific group order
72          uint transferEIN; // To record EIN of the user to whom trasnfer is inititated
73          uint transferIndex; // To record the transfer specific index of the transferee
74
75          // File Transfer Properties
76          mapping (uint => uint) transferHistory; // To maintain histroy of transfer of all EIN
77      }
```

This is a struct called “File” which actually stores the information about the File. The members of the struct includes:

i. IceGlobal.GlobalRecord rec: store the association in global record. GlobalRecord is a struct in the IceGlobal.sol
ii. bytes protocolMeta: store metadata of the protocol
iii. FileMeta fileMeta: store metadata associated with file
iv. File Properties - Encryption Properties mapping (address => bytes32) encryptedHash; // Maps Individual address to the stored hash
v. uint associatedGroupIndex: to store the group index of the group that holds the file
vi. uint associatedGroupFileIndex: to store the mapping of file in the specific group order
vii. uint transferEIN: to record EIN of the user to whom trasnfer is inititated
viii. uint transferIndex: to record the transfer specific index of the transferee
ix. mapping (uint => uint) transferHistory; // To maintain histroy of transfer of all EIN

Group

```
82      struct Group {
83          IceGlobal.GlobalRecord rec; // store the association in global record
84
85          string name; // the name of the Group
86
87          mapping (uint => IceSort.SortOrder) groupFilesOrder; // the order of files in the current group
88          uint groupFilesCount; // To keep the count of group files
89      }
```

This is a struct called “Group” which the main purpose is to sort the files in linear group like a folder. 0 or default groupID is root. The members of the struct includes:

i. IceGlobal.GlobalRecord rec: store the association in global record.

ii. string name: the name of the Group

iii. mapping (uint => IceSort.SortOrder) groupFilesOrder: the order of files in the current group

iv. uint groupFilesCount: to keep the count of group files

FUNCTION

1. FILE FUNCTION

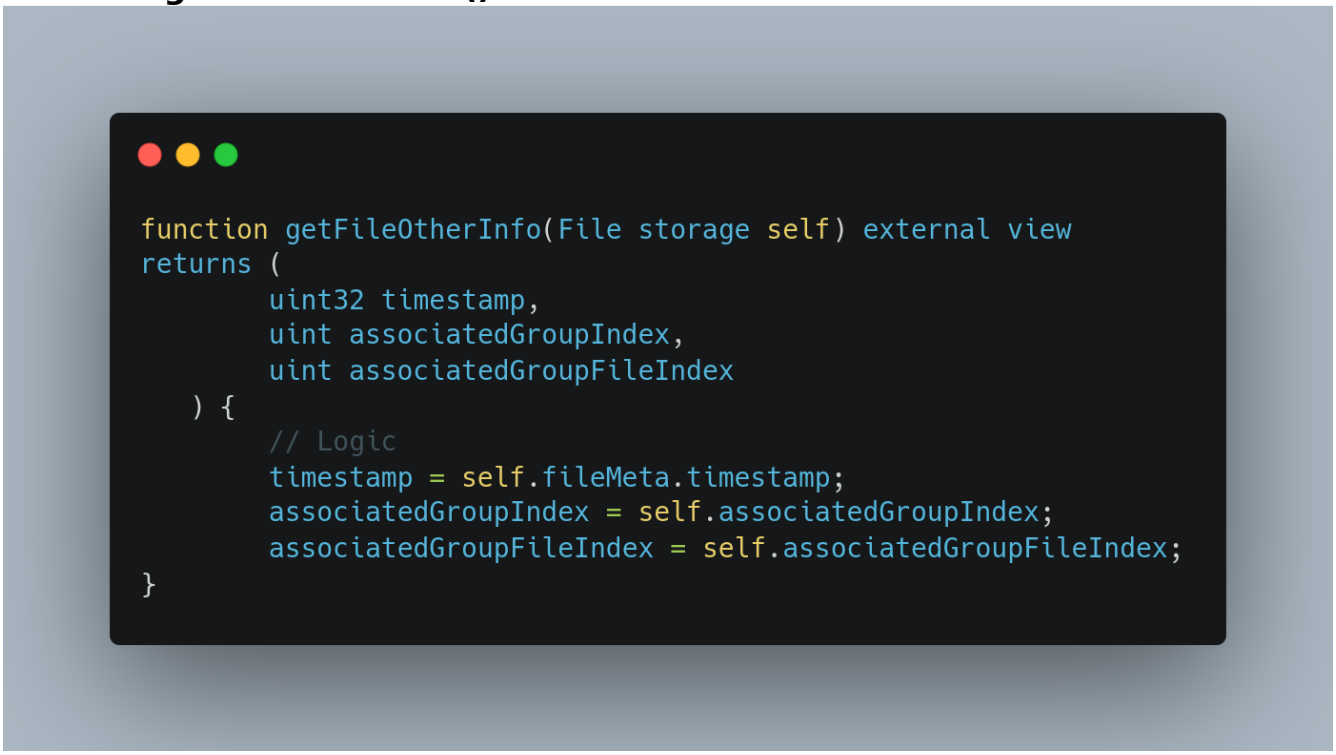
function getFileInfo():

```
function getFileInfo(File storage self) external view returns (
        uint8 protocol,
        bytes memory protocolMeta,
        string memory fileName,
        bytes32 fileHash,
        bytes22 hashExtraInfo,
        uint8 hashFunction,
        uint8 hashSize,
        bool encryptedStatus
    ) {
        // Logic
        protocol = self.fileMeta.protocol; // Protocol
        protocolMeta = self.protocolMeta; // Protocol meta

        fileName = bytes32ToString(self.fileMeta.name); // File Name, convert from byte32 to string

        fileHash = self.fileMeta.hash; // hash of the file
        hashExtraInfo = self.fileMeta.hashExtraInfo; // extra info of hash of the file (to utilize 22 bytes of wasted space)
        hashFunction = self.fileMeta.hashFunction; // the hash function used to store the file
        hashSize = self.fileMeta.hashSize; // The length of the digest

        encryptedStatus = self.fileMeta.encrypted; // Whether the file is encrypted or not
    }
```

the function visibility is external. It is a view function. It takes in a parameter “self” with a type of “File” Struct. This function is used to get the file information of an EIN (a particular user) like protocol, protocolMeta, file name, file hash, extra

info of hash, the function to store the hash, size of the hash and the status of the encryption.

function getFileOtherInfo():



```
function getFileOtherInfo(File storage self) external view
returns (
        uint32 timestamp,
        uint associatedGroupIndex,
        uint associatedGroupFileIndex
    ) {
        // Logic
        timestamp = self.fileMeta.timestamp;
        associatedGroupIndex = self.associatedGroupIndex;
        associatedGroupFileIndex = self.associatedGroupFileIndex;
}
```

this function used to get more information about an EIN that were not gotten in the previous function like timestamp of the file, group in which the file is associated to in the user's FMS. The function visibility is external. It is a view function. It takes in a parameter "self" with a type of "File" Struct.

function getFileTransferInfo():

```
function getFileTransferInfo(File storage self) external view returns (
        uint transferCount,
        uint transferEIN,
        uint transferIndex,
        bool markedForTransfer
    ) {
        // Logic
        transferCount = self.fileMeta.transferCount;
        transferEIN = self.transferEIN;
        transferIndex = self.transferIndex;
        markedForTransfer = self.fileMeta.markedForTransfer;
}
```

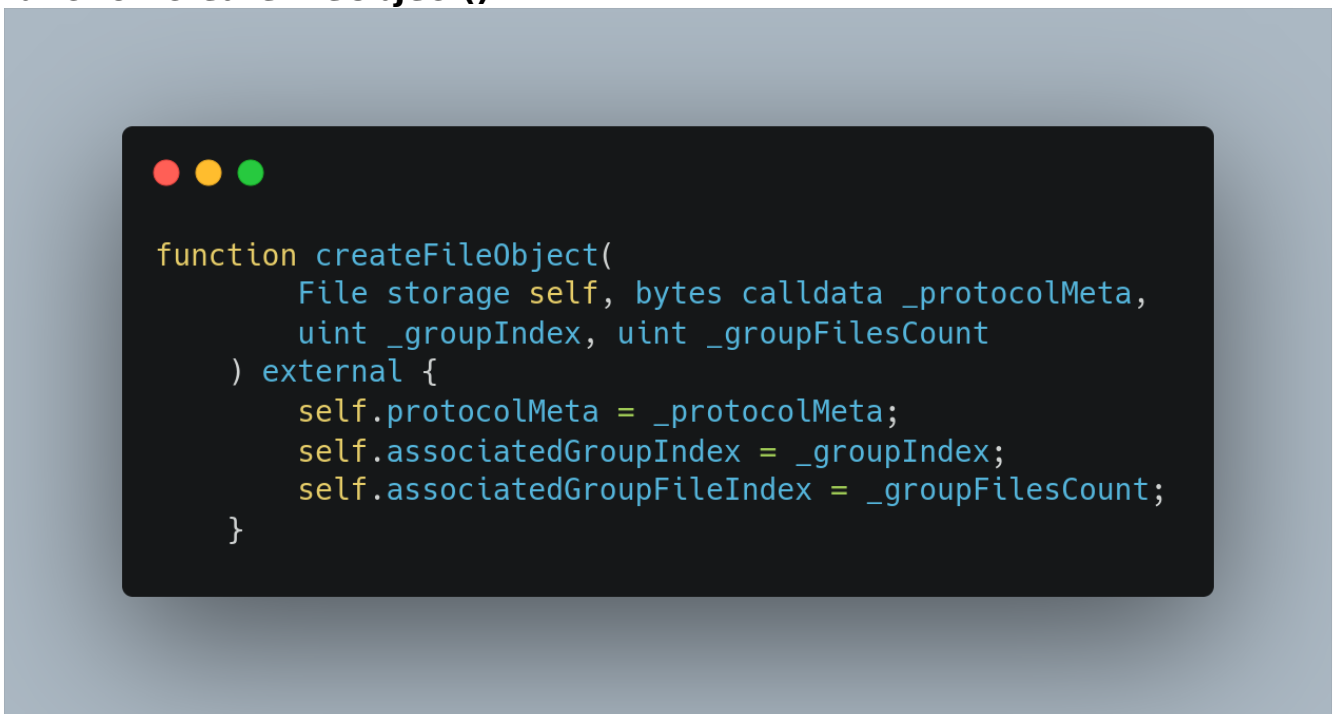
the function is used to get the transfer info of a file for an EIN. It returns the number of times the file has been transferred, the EIN of the user in which the file is currently scheduled for transfer, the transfer index of the target EIN where the file is currently mapped, and indicates if the file is marked for transfer or not. The function visibility is external. It is a view function. It takes in a parameter “self” with a type of “File” Struct.

function getFileTransferOwners():

```
function getFileTransferOwners(File storage self, uint _transferIndex) external view
    returns (uint previousOwnerEIN) {
        previousOwnerEIN = self.transferHistory[_transferIndex];
    }
```

this function is used to get the file transfer info of an EIN. It takes in two parameters: “self” with a type of “File” Struct, and _transferIndex. It returns the EIN of the user who originally owned that file. The function visibility is external. It is a view function.

function createFileObject():



```
function createFileObject(
        File storage self, bytes calldata _protocolMeta,
        uint _groupIndex, uint _groupFilesCount
    ) external {
        self.protocolMeta = _protocolMeta;
        self.associatedGroupIndex = _groupIndex;
        self.associatedGroupFileIndex = _groupFilesCount;
    }
```

the function is used to create a basic file object for a given file. It takes in four parameters: “self” with a type of “File” Struct with a location of storage, _protocolMeta which is bytes, _groupIndex and _groupFilesCount.

function createFileMetaObject(): this function is used to create a File Meta Object and attach it to File Struct

```
function createFileMetaObject(
        File storage self, uint8 _protocol, bytes32 _name, bytes32 _hash,
        bytes22 _hashExtraInfo, uint8 _hashFunction, uint8 _hashSize, bool _encrypted
    ) external {
        self.fileMeta = FileMeta(
            _name, _hash, _hashExtraInfo, _encrypted,
            false, _protocol, 1, _hashSize, uint32(now)
        );
    }
```

function writeFile(): it is used to write file to a user FMS and returns the updated file count. It calls a function **addFileToGroup** used to add file to a group, and **addToSortOrder function** used to facilitate adding of double linked list used to preserve order and form cicular linked list and the file was imported from IceSort.sol

```
function writeFile(
        File storage self, Group storage group, uint _groupIndex,
        mapping(uint => IceSort.SortOrder) storage _userFileOrderMapping,
        uint _maxFileIndex, uint _nextIndex, uint _transferEin, bytes32 _encryptedHash
    ) external returns (uint newFileCount) {
        (self.associatedGroupIndex, self.associatedGroupFileIndex) = addFileToGroup(group, _groupIndex,
_nextIndex);
        self.encryptedHash[msg.sender] = _encryptedHash;
        self.transferHistory[0] = _transferEin;
        newFileCount = _userFileOrderMapping.addToSortOrder(_userFileOrderMapping[0].prev,
_maxFileIndex, 0);
    }
```

function moveFileToGroup(): this function is used to move file from one group to another and returns the group file index using a function called inside this function block called **remapFileToGroup()** that helps to remap file from one group to another.
First, the function check if the new group is valid using a function called “condVaidSortOrder()” from IceSort.sol to check that the group order is valid. It also check whether a file has been stamped or not, an unstamped file can’t be moved.

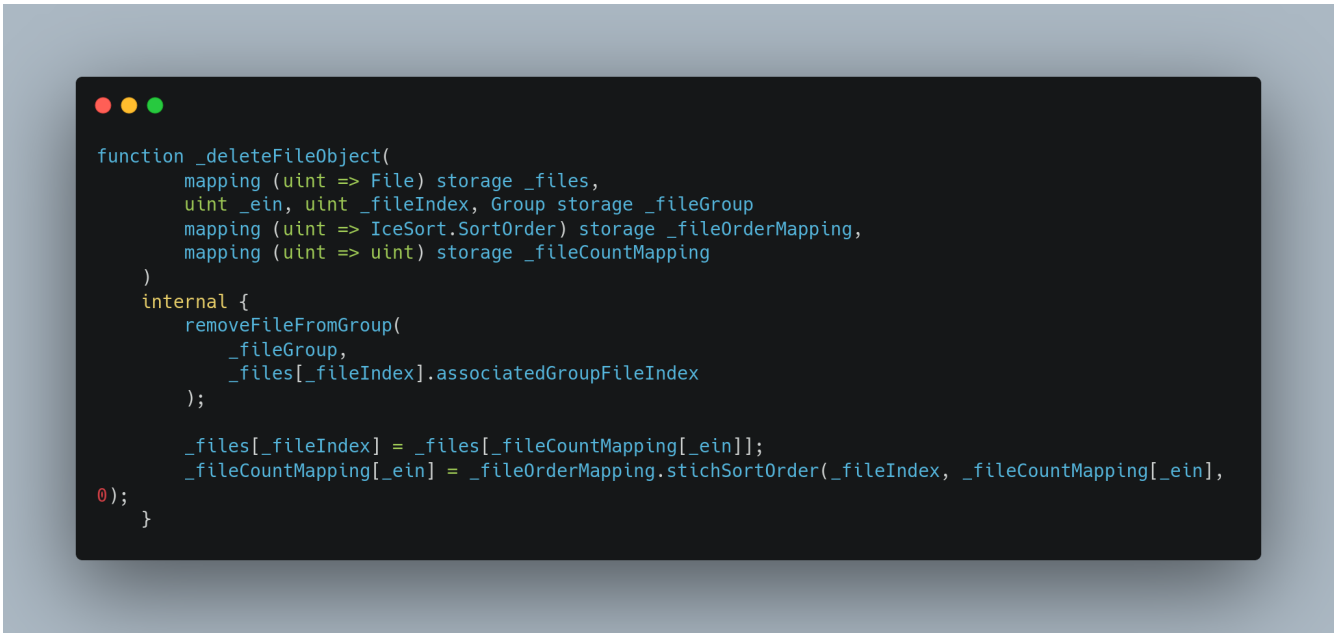
```
function moveFileToGroup(
        File storage self, uint _fileIndex, uint _newGroupIndex,
        mapping(uint => IceFMS.Group) storage _groupMapping,
        mapping(uint => IceSort.SortOrder) storage _groupOrderMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) external returns (uint groupFileIndex){
        _groupOrderMapping[_newGroupIndex].condValidSortOrder(_newGroupIndex);
        self.rec.getGlobalItemViaRecord(_globalItems).condUnstampedItem();
_groupMapping[self.associatedGroupIndex].rec.getGlobalItemViaRecord(_globalItems).condUnstampedItem();
        _groupMapping[_newGroupIndex].rec.getGlobalItemViaRecord(_globalItems).condUnstampedItem();
        groupFileIndex = remapFileToGroup(
            self,
            _fileIndex,
            _groupMapping[self.associatedGroupIndex],
            _groupMapping[_newGroupIndex],
            _newGroupIndex
        );
    }
```

_deleteFileMappings();

```
function _deleteFileMappings(
        uint _ein,
        IceGlobal.Association storage _globalItemIndividual,
        mapping (uint => mapping(uint => IceGlobal.GlobalRecord)) storage _totalSharesMapping,
        mapping (uint => mapping(uint => IceSort.SortOrder)) storage _totalShareOrderMapping,
        mapping (uint => uint) storage _shareCountMapping
    ) internal {
        _totalSharesMapping.removeAllShares(
            _ein,
            _globalItemIndividual,
            _totalShareOrderMapping,
            _shareCountMapping
        );
        _globalItemIndividual.deleteGlobalRecord();
    }
```

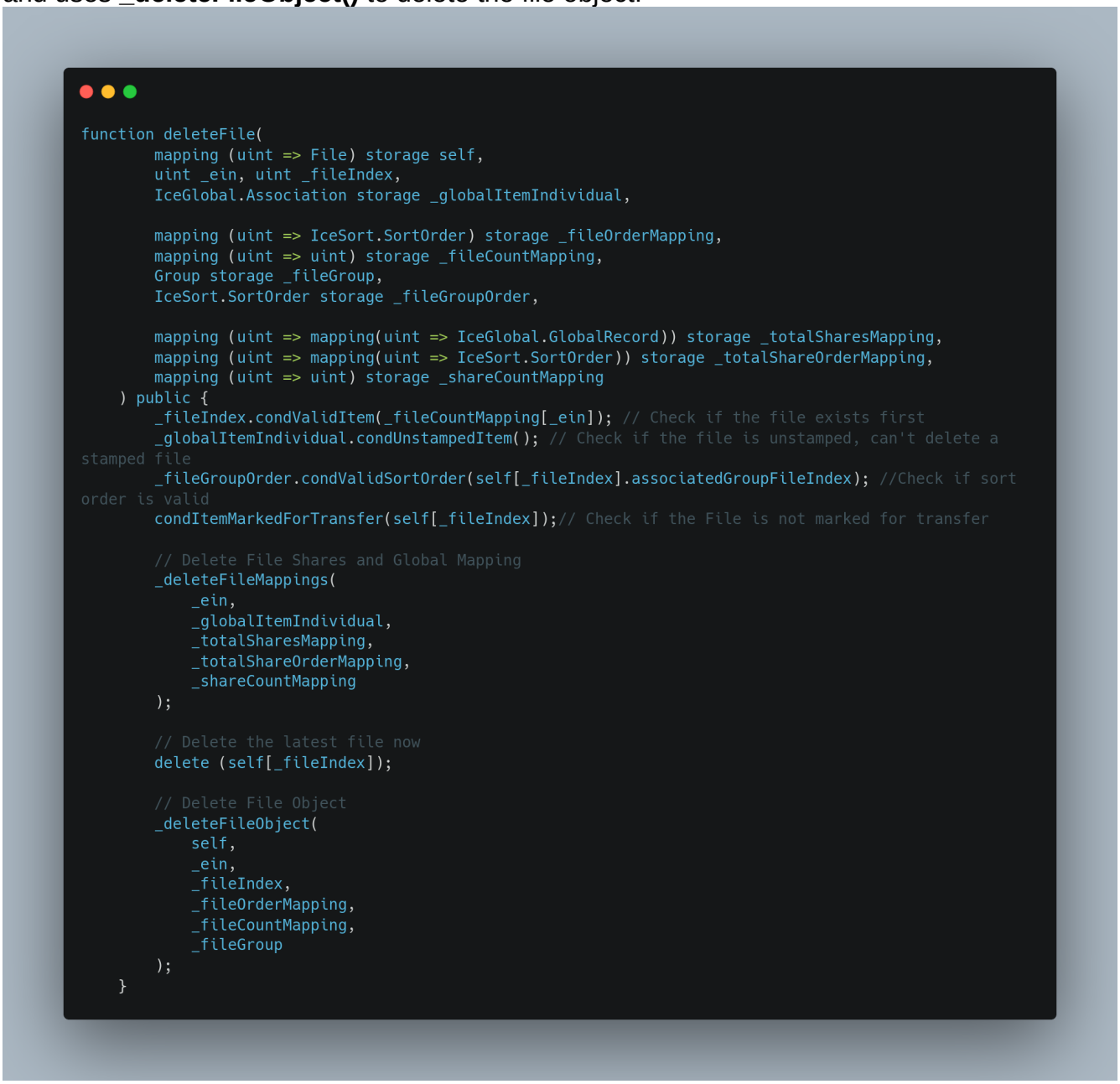
this is an intyernal function that is used to delete file mapping from a user's file management system. It first remove all item shared to other used then later remove that file from the global record.

_deleteFileObject(): is an internal function used to delete the file object. It first remove the file from the group that holds the file



```
function _deleteFileObject(
        mapping (uint => File) storage _files,
        uint _ein, uint _fileIndex, Group storage _fileGroup
        mapping (uint => IceSort.SortOrder) storage _fileOrderMapping,
        mapping (uint => uint) storage _fileCountMapping
    )
    internal {
        removeFileFromGroup(
            _fileGroup,
            _files[_fileIndex].associatedGroupFileIndex
        );

        _files[_fileIndex] = _files[_fileCountMapping[_ein]];
        _fileCountMapping[_ein] = _fileOrderMapping.stichSortOrder(_fileIndex, _fileCountMapping[_ein],
0);
    }
```

function deleteFile(): this function deletes the file of a particular owner. It uses **_deleteFileMappings()**; which is an internal function to delete the file from global mapping and uses **_deleteFileObject()** to delete the file object.



```
function deleteFile(
        mapping (uint => File) storage self,
        uint _ein, uint _fileIndex,
        IceGlobal.Association storage _globalItemIndividual,

        mapping (uint => IceSort.SortOrder) storage _fileOrderMapping,
        mapping (uint => uint) storage _fileCountMapping,
        Group storage _fileGroup,
        IceSort.SortOrder storage _fileGroupOrder,

        mapping (uint => mapping(uint => IceGlobal.GlobalRecord)) storage _totalSharesMapping,
        mapping (uint => mapping(uint => IceSort.SortOrder)) storage _totalShareOrderMapping,
        mapping (uint => uint) storage _shareCountMapping
    ) public {
        _fileIndex.condValidItem(_fileCountMapping[_ein]); // Check if the file exists first
        _globalItemIndividual.condUnstampedItem(); // Check if the file is unstamped, can't delete a
stamped file
        _fileGroupOrder.condValidSortOrder(self[_fileIndex].associatedGroupFileIndex); //Check if sort
order is valid
        condItemMarkedForTransfer(self[_fileIndex]);// Check if the File is not marked for transfer

        // Delete File Shares and Global Mapping
        _deleteFileMappings(
            _ein,
            _globalItemIndividual,
            _totalSharesMapping,
            _totalShareOrderMapping,
            _shareCountMapping
        );

        // Delete the latest file now
        delete (self[_fileIndex]);

        // Delete File Object
        _deleteFileObject(
            self,
            _ein,
            _fileIndex,
            _fileOrderMapping,
            _fileCountMapping,
            _fileGroup
        );
    }
```

2. FILE TO GROUP FUNCTION

function addFileToGroup(): this is a function used to add file to a group by using the index of the file to add it and then map group index to the group order. It takes in three parameters; self, which is a pointer to the group Struct, the group index and the file index. Then it returns associatedGroupIndex and associatedGroupFileIndex.

```
function addFileToGroup(Group storage self, int _groupIndex,uint _fileIndex
    ) public returns (uint associatedGroupIndex, uint associatedGroupFileIndex) {
        uint currentIndex = self.groupFilesCount;
        self.groupFilesCount = self.groupFilesOrder.addToSortOrder(self.groupFilesOrder[0].prev,
currentIndex, _fileIndex);

        associatedGroupIndex = _groupIndex;
        associatedGroupFileIndex = self.groupFilesCount;
    }
```

Function removeFileFromGroup(): is a function used to remove file from a group by using the groupFileOrderIndex of the file to remove it. It takes in two parameters; self, which is a pointer to the group Struct, and the groupFileOrderIndex.

```
 function removeFileFromGroup(Group storage self, uint _groupFileOrderIndex
    ) public {
        uint maxIndex = self.groupFilesCount;
        uint pointerID = self.groupFilesOrder[maxIndex].pointerID;

        self.groupFilesCount = self.groupFilesOrder.stichSortOrder(_groupFileOrderIndex, maxIndex,
pointerID);
    }
```

function remapFileToGroup(); this function is used to remap file from one group to another. It takes in five parameters and return the groupFileIndex which is a uint.

```
function remapFileToGroup(
       File storage self, uint _existingFileIndex, Group storage _oldGroup,
       Group storage _newGroup, uint _newGroupIndex
   ) public returns (uint groupFileIndex) {
       removeFileFromGroup(_oldGroup, self.associatedGroupFileIndex);

       (self.associatedGroupIndex, self.associatedGroupFileIndex) = addFileToGroup(_newGroup,
_newGroupIndex, _existingFileIndex);

       groupFileIndex = self.associatedGroupFileIndex;
   }
```

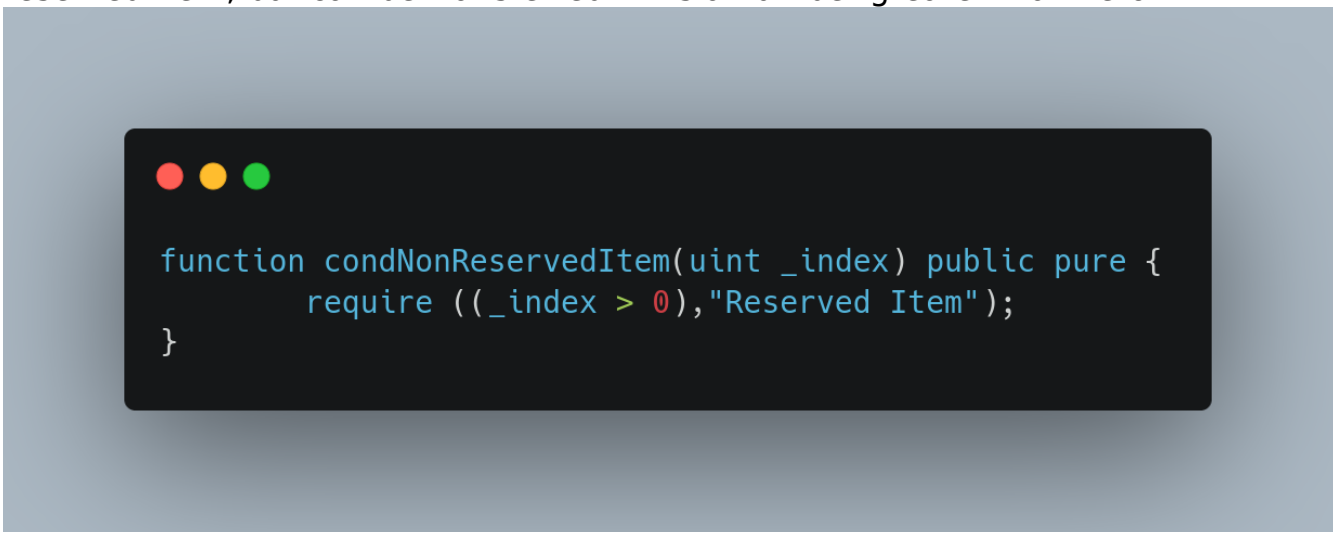
function condItemMarkedForTransfer(): is a function that is used to check if a file has been marked for transfer. The visibility of the function is *public* and it is a view function. It checks if the group file exist or not.

```
function condItemMarkedForTransfer(File storage self) public view {
        require ((self.fileMeta.markedForTransfer == false), "File already marked for Transfer");
}
```

function condMarkedForTransferee(); to check if the user you want to transfer to is eligible to receive the file.

```
function condMarkedForTransferee(File storage self, uint _transfereeEIN) public view {
        require ((self.transferEIN == _transfereeEIN), "File not marked for Transfers");
}
```

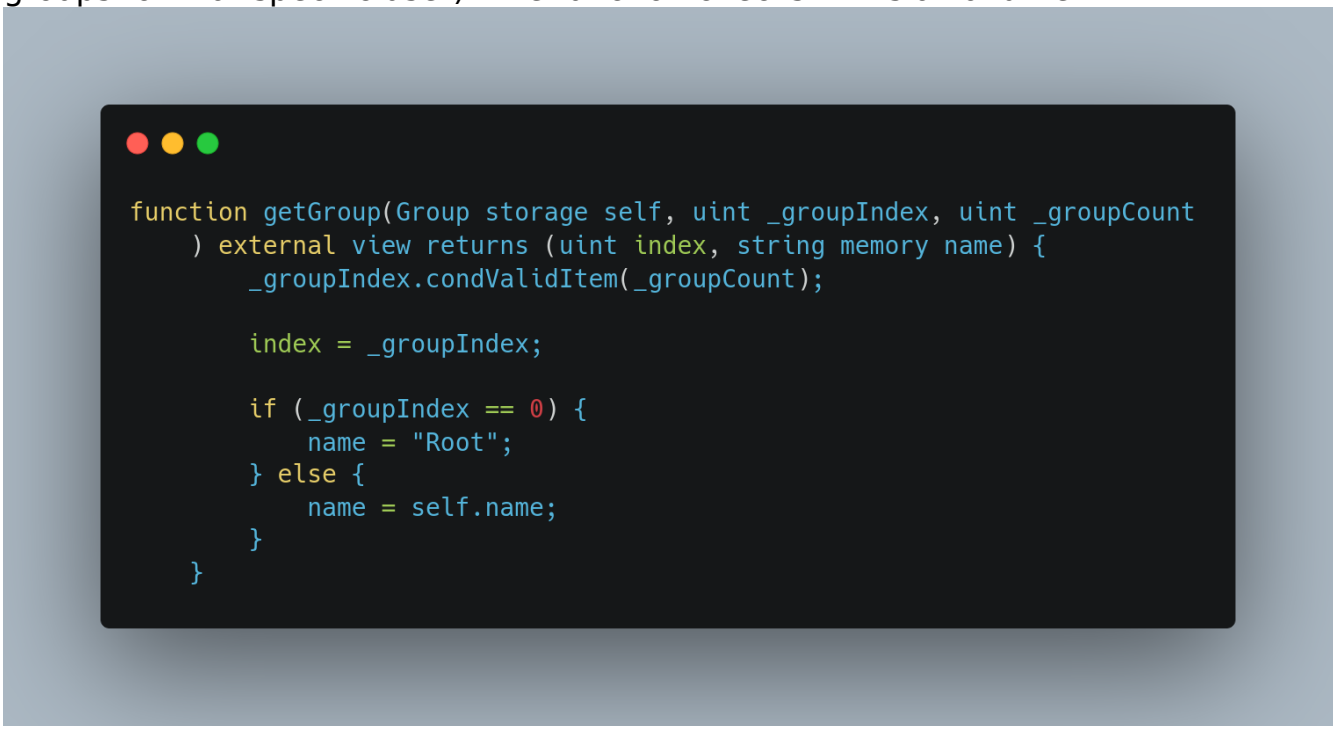
function condNonReservedItem(): to check if the file that you want to transfer is not a modified item. The visibility of the function is *public*, and it is a view function. It takes in the a parameter; index. If the index is zero, then it is a reserved item, but can be transferred if it's a number greater than zero



```
function condNonReservedItem(uint _index) public pure {
        require ((_index > 0),"Reserved Item");
}
```

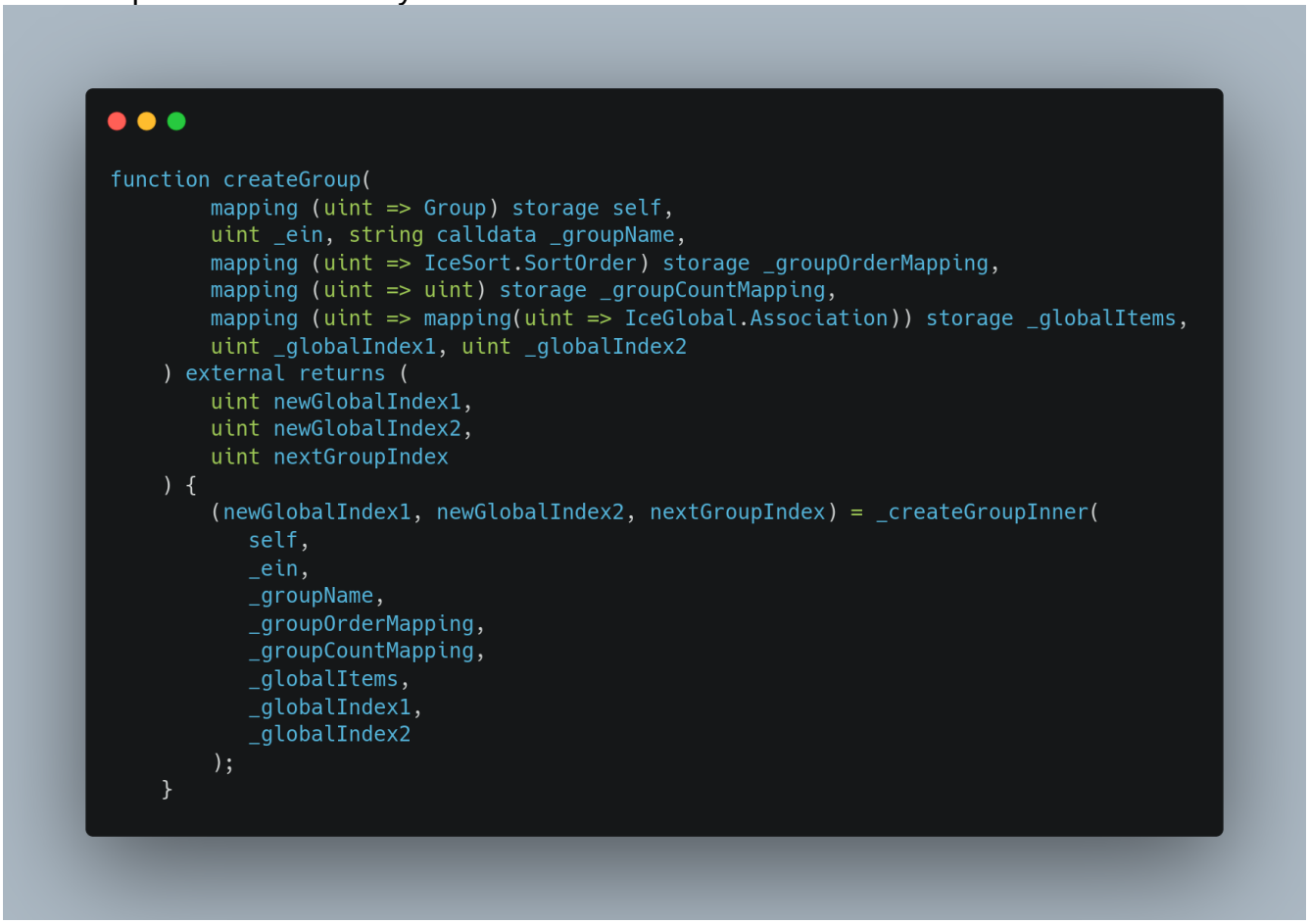
3. GROUP FUNCTION

function getGroup(): this function returns information about a particular group of files. The visibility of the function is *external* and it is a view function. It returns index and name of the group. It takes in three parameters; self, which is a pointer to the group Struct, the group index and the group count (count of the number of groups for that specific user). The function checks if it is a valid item.



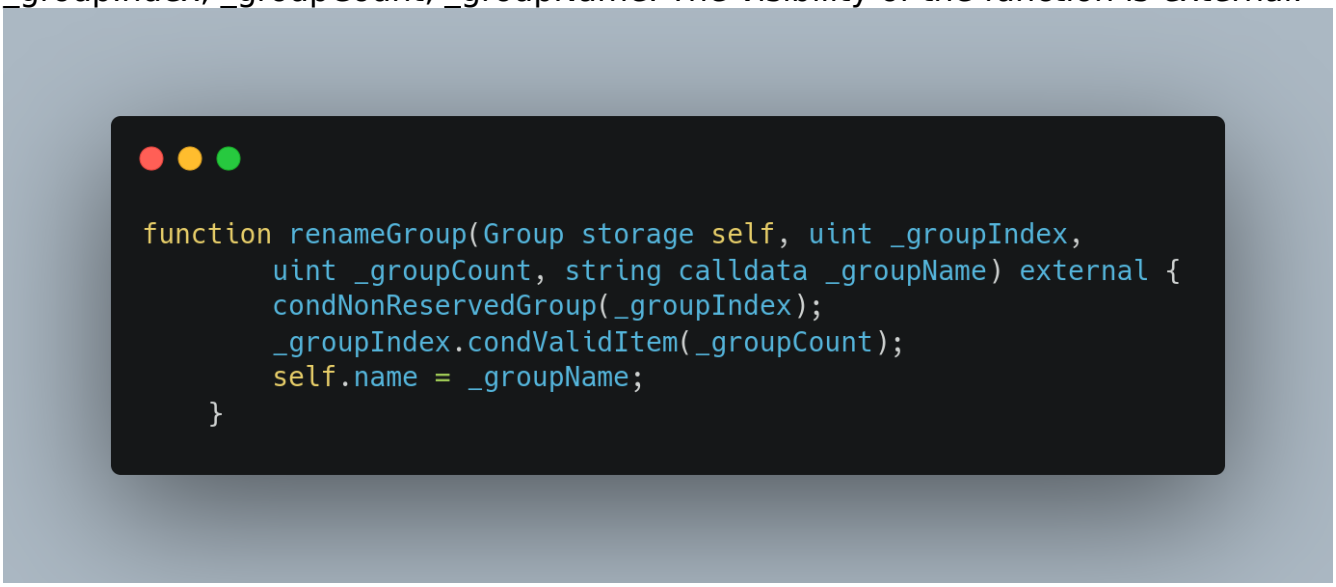
```
function getGroup(Group storage self, uint _groupIndex, uint _groupCount
    ) external view returns (uint index, string memory name) {
        _groupIndex.condValidItem(_groupCount);

        index = _groupIndex;

        if (_groupIndex == 0) {
            name = "Root";
        } else {
            name = self.name;
        }
    }
```

function createGroup(): this function is used to create a new group for the user. It takes in eight parameters and returns newGlobalIndex1, newGlobalIndex2 and nextGroupIndex. The visibility of the function is *external*.



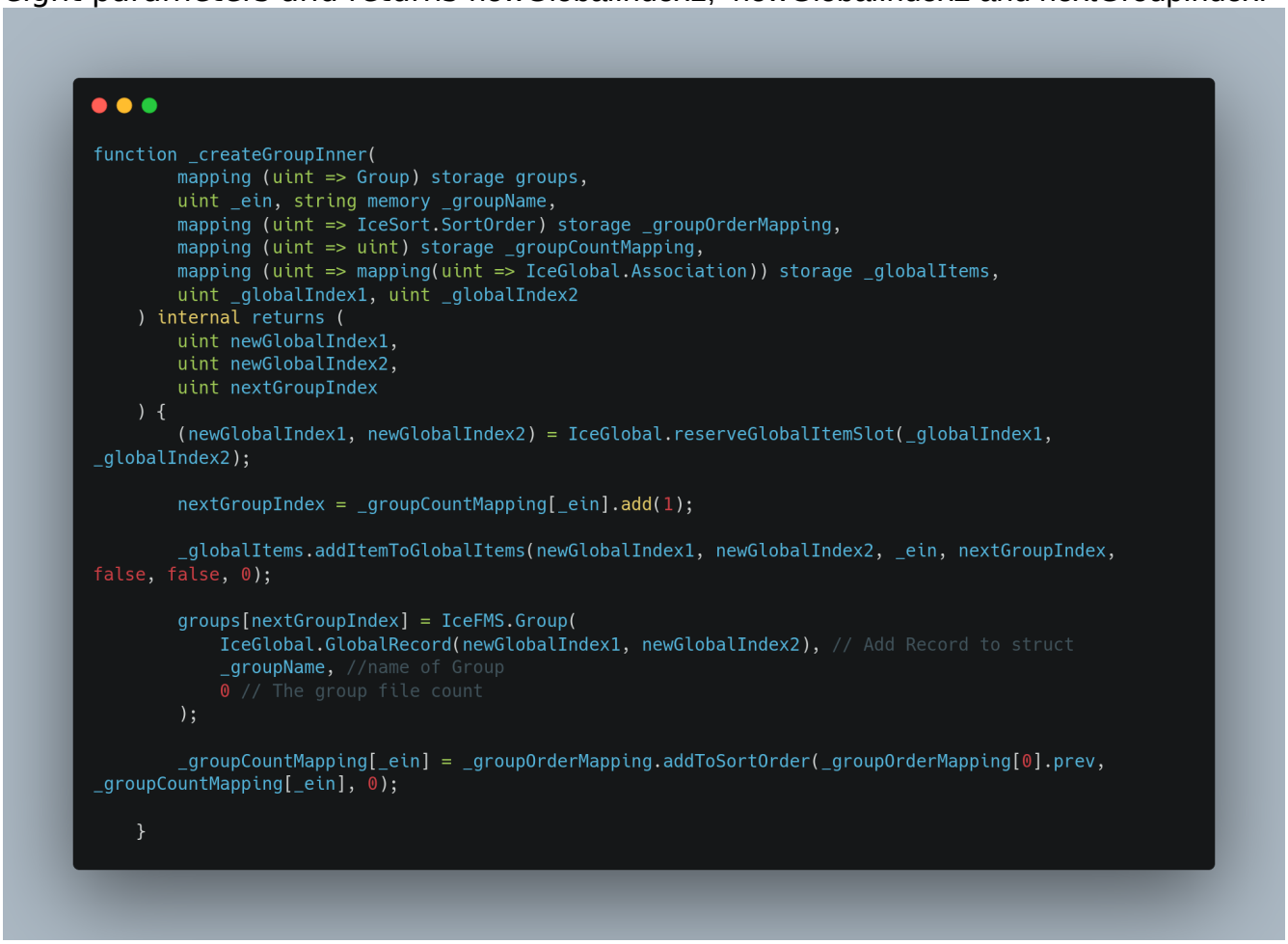
```
function createGroup(
        mapping (uint => Group) storage self,
        uint _ein, string calldata _groupName,
        mapping (uint => IceSort.SortOrder) storage _groupOrderMapping,
        mapping (uint => uint) storage _groupCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems,
        uint _globalIndex1, uint _globalIndex2
    ) external returns (
        uint newGlobalIndex1,
        uint newGlobalIndex2,
        uint nextGroupIndex
    ) {
        (newGlobalIndex1, newGlobalIndex2, nextGroupIndex) = _createGroupInner(
           self,
           _ein,
           _groupName,
           _groupOrderMapping,
           _groupCountMapping,
           _globalItems,
           _globalIndex1,
           _globalIndex2
        );
    }
```

function renameGroup(): this is a function that is to rename an existing group for the user/ein. It takes in four parameters; self, a pointer to the Group Struct, _groupIndex, _groupCount, _groupName. The visibility of the function is *external.*



```
function renameGroup(Group storage self, uint _groupIndex,
        uint _groupCount, string calldata _groupName) external {
        condNonReservedGroup(_groupIndex);
        _groupIndex.condValidItem(_groupCount);
        self.name = _groupName;
    }
```

function _createGroupInner(): this is an internal function used by **createGroup()** function to facilitate in creating new Group for the user, it takes in eight parameters and returns newGlobalIndex1, newGlobalIndex2 and nextGroupIndex.

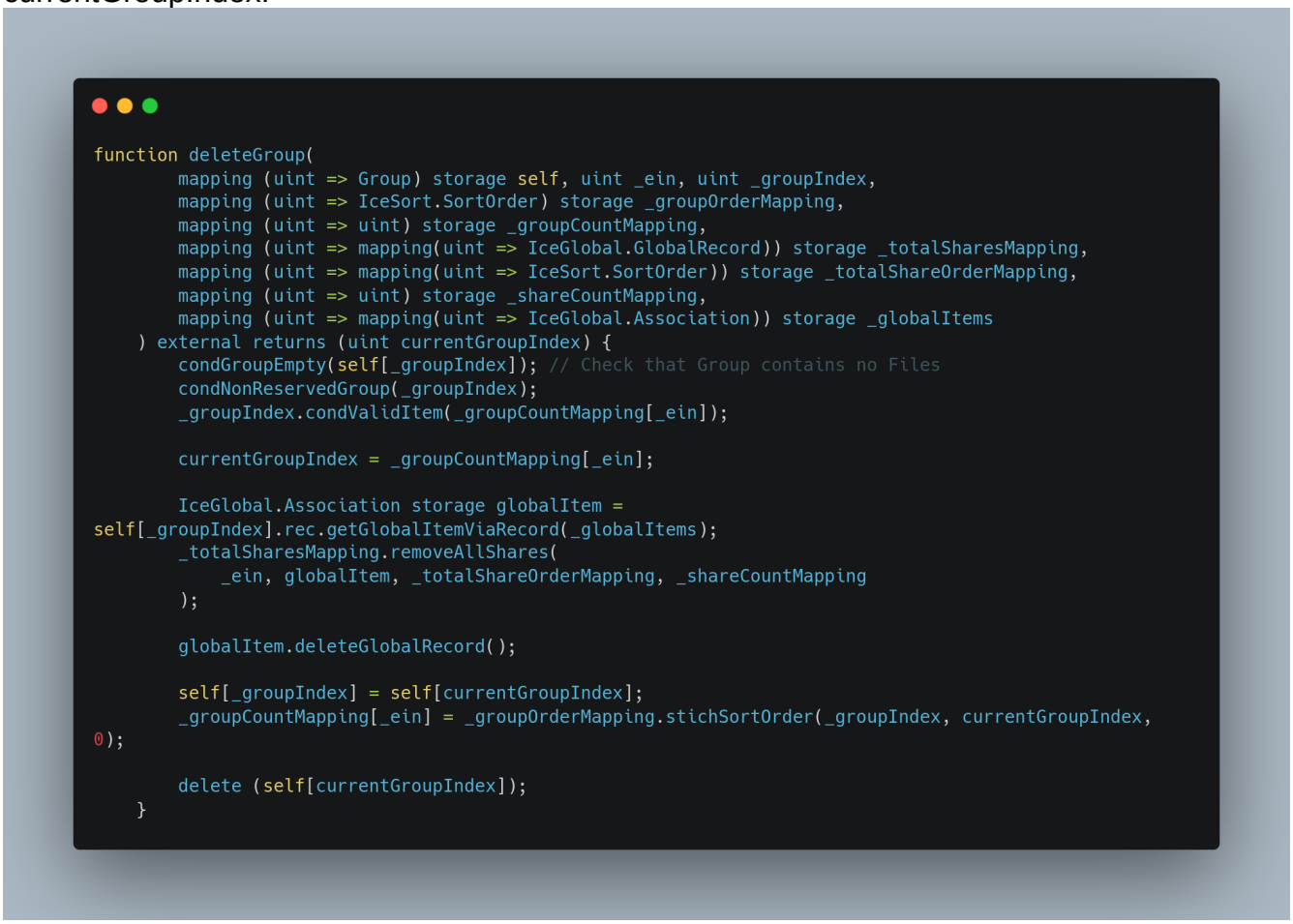


```
function _createGroupInner(
        mapping (uint => Group) storage groups,
        uint _ein, string memory _groupName,
        mapping (uint => IceSort.SortOrder) storage _groupOrderMapping,
        mapping (uint => uint) storage _groupCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems,
        uint _globalIndex1, uint _globalIndex2
    ) internal returns (
        uint newGlobalIndex1,
        uint newGlobalIndex2,
        uint nextGroupIndex
    ) {
        (newGlobalIndex1, newGlobalIndex2) = IceGlobal.reserveGlobalItemSlot(_globalIndex1,
_globalIndex2);

        nextGroupIndex = _groupCountMapping[_ein].add(1);

        _globalItems.addItemToGlobalItems(newGlobalIndex1, newGlobalIndex2, _ein, nextGroupIndex,
false, false, 0);

        groups[nextGroupIndex] = IceFMS.Group(
            IceGlobal.GlobalRecord(newGlobalIndex1, newGlobalIndex2), // Add Record to struct
            _groupName, //name of Group
            0 // The group file count
        );

        _groupCountMapping[_ein] = _groupOrderMapping.addToSortOrder(_groupOrderMapping[0].prev,
_groupCountMapping[_ein], 0);

    }
```

function condGroupEmpty(): it is a function that check that Group (the struct) Order is valied. The visibility of the function is *public*. It takes in a parameters self, a pointer to the Group Struct.

```
function condGroupEmpty(Group storage self) public view {
    require ((self.groupFilesCount == 0), "Group has Files");
}
```

function deleteGroup(): this function is used to delete an existing group for the user/ein. It takes in nine parameters. The visibility of the function is *external*. It returns currentGroupIndex.



```
function deleteGroup(
        mapping (uint => Group) storage self, uint _ein, uint _groupIndex,
        mapping (uint => IceSort.SortOrder) storage _groupOrderMapping,
        mapping (uint => uint) storage _groupCountMapping,
        mapping (uint => mapping(uint => IceGlobal.GlobalRecord)) storage _totalSharesMapping,
        mapping (uint => mapping(uint => IceSort.SortOrder)) storage _totalShareOrderMapping,
        mapping (uint => uint) storage _shareCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) external returns (uint currentGroupIndex) {
        condGroupEmpty(self[_groupIndex]); // Check that Group contains no Files
        condNonReservedGroup(_groupIndex);
        _groupIndex.condValidItem(_groupCountMapping[_ein]);

        currentGroupIndex = _groupCountMapping[_ein];

        IceGlobal.Association storage globalItem =
self[_groupIndex].rec.getGlobalItemViaRecord(_globalItems);
        _totalSharesMapping.removeAllShares(
            _ein, globalItem, _totalShareOrderMapping, _shareCountMapping
        );

        globalItem.deleteGlobalRecord();

        self[_groupIndex] = self[currentGroupIndex];
        _groupCountMapping[_ein] = _groupOrderMapping.stichSortOrder(_groupIndex, currentGroupIndex,
0);

        delete (self[currentGroupIndex]);
    }
```

function condNonReservedGroup(): the function takes in a parameter; _index. If the _index is zero, then it is a reserved item, but can be transferred if it’s a number greater than zero. The visibility of the function is *internal*, and it is a pure function.

```
function condNonReservedGroup(uint _index) internal pure {
    require ((_index > 0), "Reserved Item");
}
```

3. TRANSFER FUNCTION

function doInitiateFileTransferChecks(): this is a function to check file transfer conditions before initiating a file transfer. It takes in nine parameters. The visibility of the function is *external*, and it is a view function.

```
function doInitiateFileTransferChecks(
        mapping (uint => mapping(uint => File)) storage self,
        uint _transfererEIN, uint _transfereeEIN, uint _fileIndex,
        mapping (uint => uint) storage _fileCountMapping,
        mapping (uint => mapping(uint => Group)) storage _totalGroupsMapping,
        mapping (uint => mapping(uint => bool)) storage _blacklist,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems,
        IdentityRegistryInterface _identityRegistry
    ) external view {
        IceGlobal.condEINExists(_transfereeEIN, _identityRegistry);
        IceGlobal.condUniqueEIN(_transfererEIN, _transfereeEIN);
        _fileIndex.condValidItem(_fileCountMapping[_transfererEIN]);
        self[_transfererEIN][_fileIndex].rec.getGlobalItemViaRecord(_globalItems).condUnstampedItem();
        condItemMarkedForTransfer(self[_transfererEIN][_fileIndex]);
        _totalGroupsMapping[_transfererEIN][self[_transfererEIN]
[_fileIndex].associatedGroupIndex].rec.getGlobalItemViaRecord(_globalItems).condUnstampedItem();
        _blacklist[_transfereeEIN].condNotInList(_transfererEIN);
    }
```

function doFileTransferPart1(): this function is used to do file transfer(Part 1) from previous (current) owner to new owner. It takes in six parameters. The visibility of the function is *external*. It returns nextTransfereeIndex.

```
function doFileTransferPart1 (
        mapping (uint => mapping(uint => File)) storage self,
        uint _transfererEIN, uint _transfereeEIN, uint _fileIndex,
        mapping (uint => mapping(uint => IceSort.SortOrder)) storage _totalFilesOrderMapping,
        mapping (uint => uint) storage _fileCountMapping
    ) external returns (uint nextTransfereeIndex) {
        IceGlobal.condCheckUnderflow(_fileCountMapping[_transfererEIN]);
        uint currentTransfereeIndex = _fileCountMapping[_transfereeEIN];
        nextTransfereeIndex =  currentTransfereeIndex.add(1);
        self[_transfereeEIN][nextTransfereeIndex] = self[_transfererEIN][_fileIndex];
        uint8 tc = self[_transfereeEIN][nextTransfereeIndex].fileMeta.transferCount.add(1);
        self[_transfereeEIN][nextTransfereeIndex].transferHistory[tc] = _transfereeEIN;
        self[_transfereeEIN][nextTransfereeIndex].fileMeta.markedForTransfer = false;
        self[_transfereeEIN][nextTransfereeIndex].fileMeta.transferCount = tc;
        _fileCountMapping[_transfereeEIN] = _totalFilesOrderMapping[_transfereeEIN].addToSortOrder(
            _totalFilesOrderMapping[_transfereeEIN][0].prev,
            currentTransfereeIndex, 0
        );
    }
```

function doFileTransferPart2(): this function is used to do file transfer(Part 2) from previous (current) owner to new owner. It takes in eight parameters. The visibility of the function is *external*.

```
function doFileTransferPart2 (
        mapping (uint => mapping(uint => File)) storage self,
        uint _transfereeEIN, uint _fileIndex, uint _toRecipientGroup,
        uint _recipientGroupCount, uint _nextTransfereeIndex,
        mapping (uint => mapping(uint => Group)) storage _totalGroupsMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) external {
        _toRecipientGroup.condValidItem(_recipientGroupCount); // Check if the group exists
        (self[_transfereeEIN][_nextTransfereeIndex].associatedGroupIndex, self[_transfereeEIN]
[_nextTransfereeIndex].associatedGroupFileIndex) = addFileToGroup(
            _totalGroupsMapping[_transfereeEIN][_toRecipientGroup],
            _toRecipientGroup,
            _nextTransfereeIndex
        );

        IceGlobal.Association storage globalItem = self[_transfereeEIN]
[_fileIndex].rec.getGlobalItemViaRecord(_globalItems);

        globalItem.ownerInfo.EIN = _transfereeEIN;
        globalItem.ownerInfo.index = _nextTransfereeIndex;
    }
```

function doPermissionedFileTransfer(): this function is used to initiate requested file transfer in a permissioned manner. It takes in six parameters. The visibility of the function is external.

```
function doPermissionedFileTransfer(
        File storage self, uint _transfereeEIN,
        mapping (uint => IceGlobal.GlobalRecord) storage _transfers,
        mapping (uint => IceSort.SortOrder) storage _transferOrderMapping,
        mapping (uint => uint) storage _transferCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) external {
        _initiatePermissionedFileTransfer (
            self, _transfereeEIN, _globalItems
            _transfers, _transferOrderMapping, _transferCountMapping,
        );
    }
```

function _initiatePermissionedFileTransfer(): this function is used to initiate requested file transfer. It takes in six parameters. The visibility of the function is internal.

```
function _initiatePermissionedFileTransfer(
        File storage self, uint _transfereeEIN,
        mapping (uint => IceGlobal.GlobalRecord) storage _transfers,
        mapping (uint => IceSort.SortOrder) storage _transferOrderMapping,
        mapping (uint => uint) storage _transferCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) internal {
        self.rec.getGlobalItemViaRecord(_globalItems).condUnstampedItem();
        IceGlobal.condCheckOverflow(_transferCountMapping[_transfereeEIN]);

        uint nextTransferIndex = _transferCountMapping[_transfereeEIN] + 1;

        self.fileMeta.markedForTransfer = true;
        self.transferEIN = _transfereeEIN;
        self.transferIndex = nextTransferIndex;

        _transfers[nextTransferIndex] = self.rec;

        _transferCountMapping[_transfereeEIN] =
_transferOrderMapping.addToSortOrder(_transferOrderMapping[0].prev,
_transferCountMapping[_transfereeEIN], 0);
    }
```

function acceptFileTransferPart1(): this function is used to accept file transfer (part 1) from a user. The visibility of the function is *external.*

```
function acceptFileTransferPart1(
        mapping (uint => mapping(uint => File)) storage self,
        uint _transfererEIN, uint _transfereeEIN, uint _fileIndex
    ) external {
        condMarkedForTransferee(self[_transfererEIN][_fileIndex], _transfereeEIN);
        self[_transfererEIN][_fileIndex].fileMeta.markedForTransfer = false;
    }
```

function acceptFileTransferPart2(): this function is used to accept file transfer (part 2) from a user. The visibility of the function is external.

```
function acceptFileTransferPart2(
        mapping (uint => mapping(uint => File)) storage self,
        uint _transfereeEIN, uint _transferSpecificIndex,
        mapping (uint => IceGlobal.GlobalRecord) storage _transfersMapping,
        mapping (uint => IceSort.SortOrder) storage _transferOrderMapping,
        mapping (uint => uint) storage _transferCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) external {
        removeFileFromTransfereeMapping(
            self, _transfereeEIN, _transferSpecificIndex, _globalItems,
            _transfersMapping, _transferOrderMapping, _transferCountMapping,
        );
    }
```

function cancelFileTransfer(): this function is used to cancel file transfer inititated by the current owner and / or recipient. It takes in eight parameters. The visibility of the function is *external*.

```
function cancelFileTransfer(
        mapping (uint => mapping(uint => File)) storage self,
        uint _transfererEIN, uint _transfereeEIN, uint _fileIndex,
        mapping (uint => IceGlobal.GlobalRecord) storage _transfersMapping,
        mapping (uint => IceSort.SortOrder) storage _transferOrderMapping,
        mapping (uint => uint) storage _transferCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) external {
        condMarkedForTransferee(self[_transfererEIN][_fileIndex], _transfereeEIN);

        _cancelFileTransfer(
            self, _transfererEIN, _transfereeEIN, _fileIndex, _globalItems,
            _transfersMapping, _transferOrderMapping, _transferCountMapping
        );
    }
```

function _cancelFileTransfer(): this function to cancel file transfer inititated by the current owner or the recipient. It takes in eight parameters. The visibility of this function is *private.* This function is being called in the **cancelFileTransfer()** function.

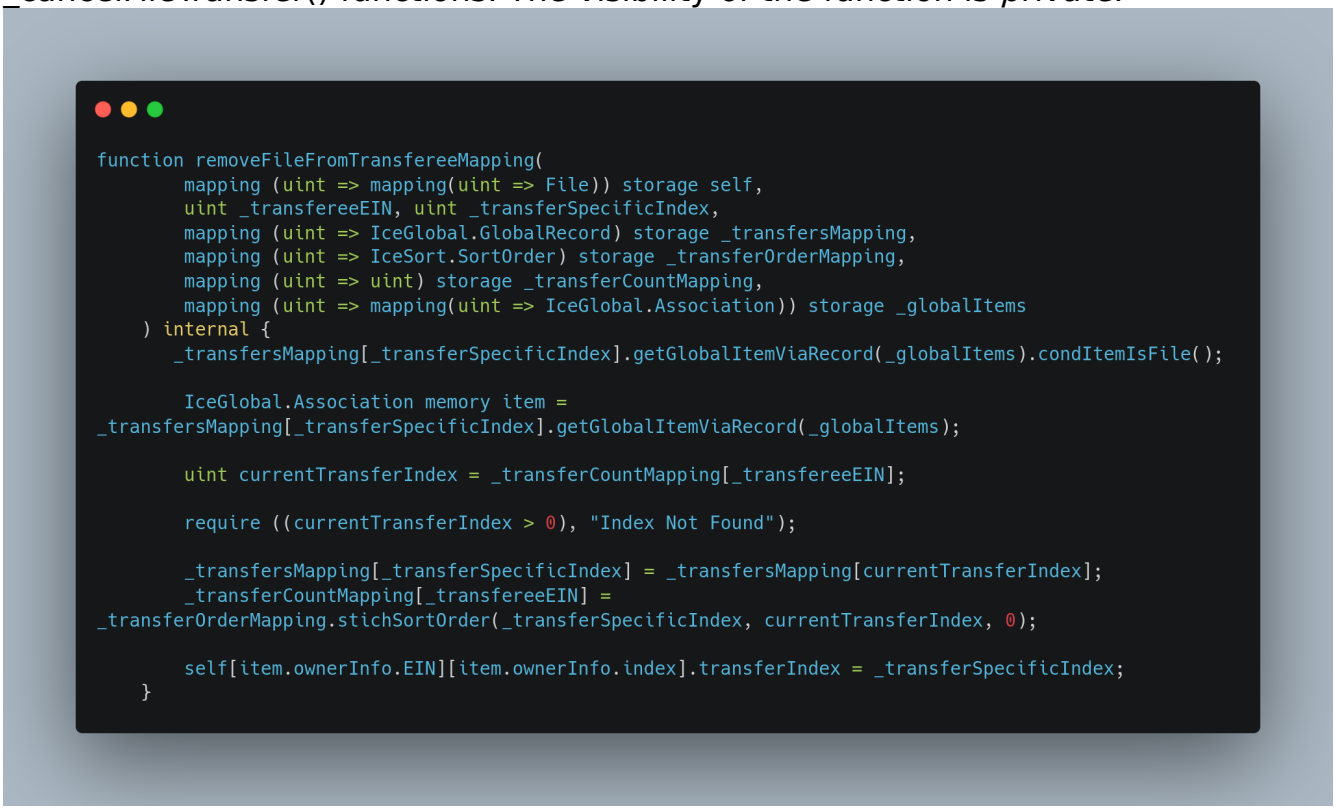
```
function _cancelFileTransfer (
        mapping (uint => mapping(uint => File)) storage self,
        uint _transfererEIN, uint _transfereeEIN, uint _fileIndex,
        mapping (uint => IceGlobal.GlobalRecord) storage _transfersMapping,
        mapping (uint => IceSort.SortOrder) storage _transferOrderMapping,
        mapping (uint => uint) storage _transferCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) private {
        self[_transfererEIN][_fileIndex].fileMeta.markedForTransfer = false;

        uint transferSpecificIndex = self[_transfererEIN][_fileIndex].transferIndex;
        removeFileFromTransfereeMapping(
            self, _transfereeEIN, transferSpecificIndex,
            _transfersMapping, _transferOrderMapping,
            _transferCountMapping, _globalItems
        );
    }
```

function stringToBytes32():

```
function stringToBytes32(string memory _source)
    public pure returns (bytes32 result) {
        bytes memory tempEmptyStringTest = bytes(_source);
        string memory tempSource = _source;

        if (tempEmptyStringTest.length == 0) return 0x0;
        assembly {
            result := mload(add(tempSource, 32))
        }
    }
```

this is a helper function to convert string to bytes32 format. The visibility of the function is public and it is a pure function. It returns result which is a bytes32. It takes in a parameter; string.

function removeFileFromTransfereeMapping(): this function is used to remove file from Transfers mapping of Transferee after file is transferred to them. It takes in eight parameters. It is used in acceptFileTransferPart2() and _cancelFileTransfer() functions. The visibility of the function is *private*.



```
function removeFileFromTransfereeMapping(
        mapping (uint => mapping(uint => File)) storage self,
        uint _transfereeEIN, uint _transferSpecificIndex,
        mapping (uint => IceGlobal.GlobalRecord) storage _transfersMapping,
        mapping (uint => IceSort.SortOrder) storage _transferOrderMapping,
        mapping (uint => uint) storage _transferCountMapping,
        mapping (uint => mapping(uint => IceGlobal.Association)) storage _globalItems
    ) internal {
       _transfersMapping[_transferSpecificIndex].getGlobalItemViaRecord(_globalItems).condItemIsFile();

        IceGlobal.Association memory item =
_transfersMapping[_transferSpecificIndex].getGlobalItemViaRecord(_globalItems);

        uint currentTransferIndex = _transferCountMapping[_transfereeEIN];

        require ((currentTransferIndex > 0), "Index Not Found");

        _transfersMapping[_transferSpecificIndex] = _transfersMapping[currentTransferIndex];
        _transferCountMapping[_transfereeEIN] =
_transferOrderMapping.stichSortOrder(_transferSpecificIndex, currentTransferIndex, 0);

        self[item.ownerInfo.EIN][item.ownerInfo.index].transferIndex = _transferSpecificIndex;
    }
```

function bytes32ToString(): this is a helper function to convert bytes32 to string format. It takes in a parameter _x which is a bytes32 and returns result which is a string.

```
function bytes32ToString(bytes32 _x)
    public pure
    returns (string memory result) {
        bytes memory bytesString = new bytes(32);
        uint charCount = 0;
        for (uint j = 0; j < 32; j++) {
            byte char = byte(bytes32(uint(_x) * 2 ** (8 * j)));
            if (char != 0) {
                bytesString[charCount] = char;
                charCount++;
            }
        }
        bytes memory bytesStringTrimmed = new bytes(charCount);
        for (uint j = 0; j < charCount; j++) {
            bytesStringTrimmed[j] = bytesString[j];
        }

        result = string(bytesStringTrimmed);
    }
```