# IThaca Market Architecture And Settlement Examples

## Ledger contract storage

Each `Ledger` contract represents a market for a specific currency pair. When a new currency pair market is needed - new Ledger contract is deployed. Within each market trades for different contractId's which use this currency pair can be made. I.e. We can trade CALL and PUT options with different expiration dates using the same Ledger contract. The storage of each Ledger contract has respective addresses of underlying and strike currency which together represent a currency pair used by this market.

```
contract Ledger {
   // ... omitted
   address public underlyingCurrency;
   address public strikeCurrency;
   // ... omitted
}
```

Within each Ledger contract client positions are represented by respective mapping. Positions are per contractId per trader. Each position denotes amount of option contracts each trader has in respective contractId (size). The size can be either negative (Sell) or positive (Buy).

```
contract Ledger {
   // ... omitted
   mapping(uint32 => mapping(address => int64)) public clientPositions;
   // ... omitted
}
```

## FundLock Storage

All data regarding actual funds/tokens of clients goes all the way down to FundLock and is updated in `Fundlock`. All clients' collateral and premium flows along with token flow from spot trades and other operations is updated in `balanceSheetS` storage mapping per user per token. This mapping is unified across all markets and trading contracts and outlines amount of funds available to client in every token used by the client for trades.

```
// client's address => tokenAddress => balanceAmount
mapping (address => mapping(address => uint256)) internal balanceSheetS;
```

## Ledger contract update flow

Orders settlement flow `updatePositions` function accepts arrays of arguments. Logically these arrays can be split onto 2 virtual data structures: "Fund Movement" and "Position". Those will be represented by array groups passed as arguments.

```
contract Ledger {
  struct PositionParam {
    uint32 contractId;
    address client;
    int64 size;
  }
  struct FundMovementParam {
      address client;
    int64 underlyingAmount;
    int64 strikeAmount;
  }
 function updatePositions(
   // Position update part of arguments
   PositionParam[] memory positions,
   // Fund Movement part of arguments
   FundMovementParam[] memory fundMovements,
   // backend tracking arg
   uint64 backendId
 ) external;
   // ... omitted
}
```

- The purpose of **Position** part is to record position information to smart contract state.
- The purpose of **FundMovement** part is to move funds in the `FundLock` without recording any information into `Ledger` state. We represent things like premiums, collaterals, spot trades, margin loan underlying transfers as `Fund Movement`.

Please also note that arrays of `positions` would not necessarily have the same length as `fundMovements` as they do represent different things. Also, it is possible for Backend to not send one part or the other (e.g. Spot Trading would only have **Fund Movements** part and arrays for positions will be empty), however smart contract will revert if both `positionClients` and `fundMovementClients` arrays are empty signifying an error on the backend or the fact that a TX is being sent with no data in it, for which we would still have to pay gas.

The `uint64 backendId` argument is meant for injecting a backend generated transaction id to be able to immediately assign it to transaction without waiting for txHash to become available. This `backendId` is always used to emit an appropriate event in `Ledger` contract with the following signature:

```
event PositionsUpdated(
   uint64 indexed backendId
);
```

Naive Matching example data

A simple example of orders and expected input of `updatePositions` is as follows:

| Order # | size | Price | Side | Strike | Option type | User | Match price | contractId |
|---|---|---|---|---|---|---|---|---|
| Order 1 | 20 | 11 | BID | 15 | CALL | User 1 | 10 | 100500 |
| Order 2 | 20 | 9 | ASK | 15 | CALL | User 2 | 10 | 100500 |

Movement of funds need to include premium information in our case. With the following representation we have premium subtracted from account of User 1 and added to account of User 2. Importantly we introduce `Moves funds in FundLock` utility column in the following 2 tables. This column is never passed to smart contract functions as argument and is present rather to denote the fact that such row would either make `FundLock` contract to move funds or not do anything. In general all non-zero rows would move funds.

In the below example, **User 1 pays premium** to User 2 for his BID/BUY and **User 2 pays collateral** to the market for his ASK/SELL.

Please note the sign of the arguments. Here positive sign means client pays and negative means he receives.

| Positions row index | client | contractId | size | Moves funds in Fundlock |
|---|---|---|---|---|
| 0 | User 1 | 100500 | 20 | no |
| 1 | User 2 | 100500 | -20 | no |

| Fund movements row index | client | underlyingAmount | strikeAmount | Moves funds in FundLock |
|---|---|---|---|---|
| 0 | User 1 | 0 | 200 | yes |
| 1 | User 2 | 20 | -200 | yes |

**Position change example data**

Current smart contract state is important for validating incoming position's data. We provide a number of examples which describe situations when Users exit from their respective positions or are swapping long position for short one and vice-versa. After these examples we summarize the rules that apply to validate the input data and explain why those validations are appropriate. All following examples assume a 2 step flow. The second step is provided in each of the respective examples. Setup data for all the cases below is shared and similar to Naive Matching example data. After Naive Matching example data is sent to the contract the state is as follows:

**Ledger State**

```
{
 clientPositions: {
   100500: {
     'User 1': 20,
     'User 2': -20,
   }
 },
}
```

**FundLock State**

```
{
 balanceSheetS: {
   'User 1': {
     'underlyingAddress': IB`,
     'strikeAddress': IB` - 200
   },
   'User 2': {
     'underlyingAddress': IB` - 20,
     'strikeAddress': IB` + 200
   }
 }
}


` IB - Initial Balance
```

Example 1. Short position partial and full exit.

The second step is when User 2 submits an order for long position with same contractId:

| Order # | size | Price | Side | Strike | Option type | User | Match price | contractId |
|---|---|---|---|---|---|---|---|---|
| Order 1 | 10 | 11 | BID | 15 | CALL | User 2 | 10 | 100500 |
| Order 2 | 10 | 9 | ASK | 15 | CALL | User 3 | 10 | 100500 |

Such submission would result in the following input data to the `updatePositions` function:

| Positions row index | client | contractId | size | Moves funds in Fundlock |
|---|---|---|---|---|
| 0 | User 2 | 100500 | 10 | no |
| 1 | User 3 | 100500 | -10 | no |

| Fund movements row index | Client | underlyingAmount | strikeAmount | Moves funds in FundLock |
|---|---|---|---|---|
| 0 | User 2 | -10 | 100 | yes |
| 1 | User 3 | 10 | -100 | yes |

After these changes are applied to state it would look as follows:

**Ledger State**

```
{
 clientPositions: {
   100500: {
     'User 1': 20,
     'User 2': -10,
     'User 3': -10
   }
 },
}
```

**FundLock State**

```
{
 balanceSheetS: {
   'User 1': {
     'underlyingAddress': IB`,
     'strikeAddress': IB` - 200
   },
   'User 2': {
     'underlyingAddress': IB` - 20 + 10,
     'strikeAddress': IB` + 200 - 100
   }
   'User 3': {
     'underlyingAddress': IB` - 10,
     'strikeAddress': IB` + 100
   }
 }
}


` IB - Initial Balance
```

As a result of this execution User 2 would get 10 underlying currency tokens back to his account and would have to pay 100 strike tokens in premium. User 3 would have to, in-turn, deposit 10 underlying currency tokens to back newly formed option.

Full exit example data is almost the same as the one provided in the above example. Distinction is only in the size of the order which would be 20. The input data for positions and the resulting state would be as follows:

| Positions row index | client | contractId | size | Moves funds in Fundlock |
|---|---|---|---|---|
| 0 | User 3 | 100500 | -20 | no |
| 1 | User 2 | 100500 | 20 | no |

```
{
 clientPositions: {
   100500: {
     'User 1': 20,
     'User 2': 0,
     'User 3': -20
   }
 },
}
```

Example 2. Long position partial and full exit.

The second step is when User 1 submits an order for short position with same contractId:

| Order # | size | Price | Side | Strike | Option type | User | Match price | contractId |
|---|---|---|---|---|---|---|---|---|

|  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| Order 1 | 10 | 11 | BID | 15 | CALL | User 3 | 10 | 100500 |
| Order 2 | 10 | 9 | ASK | 15 | CALL | User 1 | 10 | 100500 |

Such submission would result in the following input data to the `updatePositions` function:

| Positions row index | client | contractId | size | Moves funds in Fundlock |
|---|---|---|---|---|
| 0 | User 3 | 100500 | 10 | no |
| 1 | User 1 | 100500 | -10 | no |

| Fund movements row index | client | underlyingAmount | strikeAmount | Moves funds in FundLock |
|---|---|---|---|---|
| 0 | User 3 | 0 | 100 | yes |
| 1 | User 1 | 0 | -100 | yes |

After these changes are applied to state it would look as follows:

**IThaca State**

```
{
 clientPositions: {
   100500: {
     'User 1': 10,
     'User 2': -20,
     'User 3': 10
   }
 },
}
```

**FundLock State**

```
{
 balanceSheetS: {
   'User 1': {
     'underlyingAddress': IB`,
     'strikeAddress': IB` - 200 + 100
   },
   'User 2': {
     'underlyingAddress': IB` - 20,
     'strikeAddress': IB` + 200
   },
   'User 3': {
     'underlyingAddress': IB`,
     'strikeAddress': IB` - 100
   }
 }
}
```

```
` IB - Initial Balance
```

As a result of this execution User 1 would have to post 0 underlying currency tokens as collateral even though he is engaging in a short position. He would also receive 100 underlying tokens in premium. There is no collateral modification in Ledger contract when this trade executes since all necessary collateral is already provided during the first trade.

To summarize the validation which would be applied: When user has a positive position in Ledger state and engages in negative position trade as a result the absolute value for his position's state would be decreased both of the collateral values for his position row should be zero. Full exit example data is almost the same as the one provided in this example. Distinction is only in the size of the order which would be 20. The input data for positions and the resulting state would be as follows:

| Positions row index | client | contractId | size | Moves funds in Fundlock |
|---|---|---|---|---|
| 0 | User 3 | 100500 | 20 | no |
| 1 | User 1 | 100500 | -20 | no |

```
{
 clientPositions: {
   100500: {
     'User 1': 0,
     'User 2': -20,
     'User 3': 20
   }
 },
}
```

Example 3. Short position switch to long position.

The second step is when User 2 submits an order for long position with same contractId and the size of this position is larger then the size of the short position currently owned by User 2:

| Order # | size | Price | Side | Strike | Option type | User | Match price | contractId |
|---|---|---|---|---|---|---|---|---|
| Order 1 | 30 | 11 | BID | 15 | CALL | User 2 | 10 | 100500 |
| Order 2 | 30 | 9 | ASK | 15 | CALL | User 3 | 10 | 100500 |

Such submission would result in the following input data to the `updatePositions` function:

| Positions row index | client | contractId | size | Moves funds in Fundlock |
|---|---|---|---|---|
| 0 | User 2 | 100500 | 30 | no |

| | | | | |
|---|---|---|---|---|
| 1 | User 3 | 100500 | -30 | no |

| Fund movements row index | client | underlyingAmount | strikeAmount | Moves funds in FundLock |
|---|---|---|---|---|
| 0 | User 2 | -20 | 300 | yes |
| 1 | User 3 | 30 | -300 | yes |

After these changes are applied to Ledger state it would look as follows:

```
{
 clientPositions: {
   100500: {
     'User 1': 20,
     'User 2': 10,
     'User 3': -30
   }
 },
}
```

As a result of this execution User 2 would get 20 underlying currency tokens back to his account and would have to pay 300 strike tokens in premium. User 3 would have to in-turn deposit 30 underlying currency tokens to back newly formed option. User 1 and User 2 are now paired on the long side with User 3.

Example 4. Long position switch to short position.

The second step is when User 1 submits an order for short position with same contractId and the size of this position is larger then the size of the long position currently owned by User 1:

| Order # | size | Price | Side | Strike | Option type | User | Match price | contractId |
|---|---|---|---|---|---|---|---|---|
| Order 1 | 30 | 11 | BID | 15 | CALL | User 3 | 10 | 100500 |
| Order 2 | 30 | 9 | ASK | 15 | CALL | User 1 | 10 | 100500 |

Such submission would result in the following input data to the `updatePositions` function:

| Positions row index | client | contractId | size | Moves funds in Fundlock |
|---|---|---|---|---|
| 0 | User 3 | 100500 | 30 | no |
| 1 | User 1 | 100500 | -30 | no |

| Fund movements row index | client | underlyingAmount | strikeAmount | Moves funds in FundLock |
|---|---|---|---|---|
| 0 | User 3 | 0 | 300 | yes |
| 1 | User 1 | 10 | -300 | yes |

```
{
clientPositions: {
100500: {
'User 1': -10,
'User 2': -20,
'User 3': 30
}
},
}
```

## NatSpec Technical Documentation

### NatSpec documentation on each contract in the system

# AccessController

## Methods

### DEFAULT_ADMIN_ROLE

function DEFAULT_ADMIN_ROLE() external view returns (bytes32)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bytes32 | |

### checkRole

function checkRole(bytes32 role, address account) external view

#### Parameters

| Name | Type | Description |
|---|---|---|
| role | bytes32 | |
| account | address | |

### getRoleAdmin

function getRoleAdmin(bytes32 role) external view returns (bytes32)

*Returns the admin role that controls `role`. See {grantRole} and {revokeRole}. To change a role's admin, use {_setRoleAdmin}.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| role | bytes32 | |

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bytes32 | |

### grantRole

function grantRole(bytes32 role, address account) external nonpayable

*Grants `role` to `account`. If `account` had not been already granted `role`, emits a {RoleGranted} event. Requirements: - the caller must have `role`'s admin role. May emit a {RoleGranted} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| role | bytes32 | |
| account | address | |

### hasRole

function hasRole(bytes32 role, address account) external view returns (bool)

*Returns `true` if `account` has been granted `role`.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| role | bytes32 | |
| account | address | |

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bool | |

## initialize

function initialize() external nonpayable

## proxiableUUID

function proxiableUUID() external view returns (bytes32)

*Implementation of the ERC1822 {proxiableUUID} function. This returns the storage slot used by the implementation. It is used to validate the implementation's compatibility when performing an upgrade. IMPORTANT: A proxy pointing at a proxiable contract should not be considered proxiable itself, because this risks bricking a proxy that upgrades to it, by delegating to itself until out of gas. Thus it is critical that this function revert if invoked through a proxy. This is guaranteed by the `notDelegated` modifier.*

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bytes32 | |

## renounceRole

function renounceRole(bytes32 role, address account) external nonpayable

*Revokes `role` from the calling account. Roles are often managed via {grantRole} and {revokeRole}: this function's purpose is to provide a mechanism for accounts to lose their privileges if they are compromised (such as when a trusted device is misplaced). If the calling account had been revoked `role`, emits a {RoleRevoked} event. Requirements: - the caller must be `account`. May emit a {RoleRevoked} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| role | bytes32 | |
| account | address | |

## revokeRole

function revokeRole(bytes32 role, address account) external nonpayable

*Revokes `role` from `account`. If `account` had been granted `role`, emits a {RoleRevoked} event. Requirements: - the caller must have `role`'s admin role. May emit a {RoleRevoked} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|

|  |  |
|---|---|
| role | bytes32 |
| account | address |

### supportsInterface

function supportsInterface(bytes4 interfaceId) external view returns (bool)

*See {IERC165-supportsInterface}.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| interfaceId | bytes4 |  |

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bool |  |

### upgradeTo

function upgradeTo(address newImplementation) external nonpayable

*Upgrade the implementation of the proxy to `newImplementation`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address |  |

### upgradeToAndCall

function upgradeToAndCall(address newImplementation, bytes data) external payable

*Upgrade the implementation of the proxy to `newImplementation`, and subsequently execute the function call encoded in `data`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address |  |
| data | bytes |  |

## Events

### AdminChanged

event AdminChanged(address previousAdmin, address newAdmin)

#### Parameters

| Name | Type | Description |
|---|---|---|
| previousAdmin | address | |
| newAdmin | address | |

### BeaconUpgraded

event BeaconUpgraded(address indexed beacon)

#### Parameters

| Name | Type | Description |
|---|---|---|
| beacon `indexed` | address | |

### Initialized

event Initialized(uint8 version)

#### Parameters

| Name | Type | Description |
|---|---|---|
| version | uint8 | |

### RoleAdminChanged

event RoleAdminChanged(bytes32 indexed role, bytes32 indexed previousAdminRole, bytes32 indexed newAdminRole)

#### Parameters

| Name | Type | Description |
|---|---|---|
| role `indexed` | bytes32 | |
| previousAdminRole `indexed` | bytes32 | |
| newAdminRole `indexed` | bytes32 | |

### RoleGranted

event RoleGranted(bytes32 indexed role, address indexed account, address indexed sender)

#### Parameters

| Name | Type | Description |
|---|---|---|
| role `indexed` | bytes32 | |
| account `indexed` | address | |
| sender `indexed` | address | |

### RoleRevoked

event RoleRevoked(bytes32 indexed role, address indexed account, address indexed sender)

#### Parameters

| Name | Type | Description |
|---|---|---|
| role `indexed` | bytes32 | |
| account `indexed` | address | |
| sender `indexed` | address | |

### Upgraded

event Upgraded(address indexed implementation)

#### Parameters

| Name | Type | Description |
|---|---|---|
| implementation `indexed` | address | |

# FundLockExecutable

## Methods

### execute

function execute(bytes32 commandId, string sourceChain, string sourceAddress, bytes payload) external nonpayable

**Parameters**

| Name | Type | Description |
|---|---|---|
| commandId | bytes32 | |
| sourceChain | string | |
| sourceAddress | string | |
| payload | bytes | |

## executeWithToken

function executeWithToken(bytes32 commandId, string sourceChain, string sourceAddress, bytes payload, string tokenSymbol, uint256 amount) external nonpayable

**Parameters**

| Name | Type | Description |
|---|---|---|
| commandId | bytes32 | |
| sourceChain | string | |
| sourceAddress | string | |
| payload | bytes | |
| tokenSymbol | string | |
| amount | uint256 | |

## fundLock

function fundLock() external view returns (contract IFundLock)

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | contract IFundLock | |

## gateway

function gateway() external view returns (contract IAxelarGateway)

**Returns**

| Name | Type | Description |
|---|---|---|

| | |
|---|---|
| _0 | contract IAxelarGateway |

### initialize

function initialize(contract IAccessController _accessController, contract IFundLock _fundLock, contract IAxelarGateway _gateway) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| _accessController | contract IAccessController | |
| _fundLock | contract IFundLock | |
| _gateway | contract IAxelarGateway | |

### proxiableUUID

function proxiableUUID() external view returns (bytes32)

*Implementation of the ERC1822 {proxiableUUID} function. This returns the storage slot used by the implementation. It is used to validate the implementation's compatibility when performing an upgrade. IMPORTANT: A proxy pointing at a proxiable contract should not be considered proxiable itself, because this risks bricking a proxy that upgrades to it, by delegating to itself until out of gas. Thus it is critical that this function revert if invoked through a proxy. This is guaranteed by the `notDelegated` modifier.*

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bytes32 | |

### upgradeTo

function upgradeTo(address newImplementation) external nonpayable

*Upgrade the implementation of the proxy to `newImplementation`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address | |

### upgradeToAndCall

function upgradeToAndCall(address newImplementation, bytes data) external payable

*Upgrade the implementation of the proxy to `newImplementation`, and subsequently execute the function call encoded in `data`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

**Parameters**

| Name | Type | Description |
|---|---|---|
| newImplementation | address | |
| data | bytes | |

## Events

### AdminChanged

event AdminChanged(address previousAdmin, address newAdmin)

**Parameters**

| Name | Type | Description |
|---|---|---|
| previousAdmin | address | |
| newAdmin | address | |

### BeaconUpgraded

event BeaconUpgraded(address indexed beacon)

**Parameters**

| Name | Type | Description |
|---|---|---|
| beacon `indexed` | address | |

### Initialized

event Initialized(uint8 version)

**Parameters**

| Name | Type | Description |
|---|---|---|
| version | uint8 | |

### Upgraded

event Upgraded(address indexed implementation)

**Parameters**

| Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| implementation `indexed` | address | |

## Errors

### NotApprovedByGateway

error NotApprovedByGateway()

# FundLockGateway

## Methods

### deposit

function deposit(string symbol, uint256 value) external payable

Makes a cross-chain deposit to the settlement chain

#### Parameters

| Name | Type | Description |
|---|---|---|
| symbol | string | the token symbol to deposit |
| value | uint256 | the amount of the deposit |

### gasReceiver

function gasReceiver() external view returns (contract IAxelarGasService)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | contract IAxelarGasService | |

### gateway

function gateway() external view returns (contract IAxelarGateway)

public variables

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | contract IAxelarGateway | |

### initialize

function initialize(contract IAccessController _accessController, contract IAxelarGateway _gateway, contract IAxelarGasService _gasService, string _settlementChain, string _settlementContract) external nonpayable

**Parameters**

| Name | Type | Description |
|---|---|---|
| _accessController | contract IAccessController | |
| _gateway | contract IAxelarGateway | |
| _gasService | contract IAxelarGasService | |
| _settlementChain | string | |
| _settlementContract | string | |

## proxiableUUID

function proxiableUUID() external view returns (bytes32)

*Implementation of the ERC1822 {proxiableUUID} function. This returns the storage slot used by the implementation. It is used to validate the implementation's compatibility when performing an upgrade. IMPORTANT: A proxy pointing at a proxiable contract should not be considered proxiable itself, because this risks bricking a proxy that upgrades to it, by delegating to itself until out of gas. Thus it is critical that this function revert if invoked through a proxy. This is guaranteed by the `notDelegated` modifier.*

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | bytes32 | |

## release

function release(string symbol, uint256 withdrawTimestamp) external payable

Makes a cross-chain release request to the settlement chain A withdraw request must have been done first.

**Parameters**

| Name | Type | Description |
|---|---|---|
| symbol | string | the token symbol to release |
| withdrawTimestamp | uint256 | the timestamp of the withdraw request |

## settlementChain

function settlementChain() external view returns (string)

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | string | |

## settlementContract

function settlementContract() external view returns (string)

### Returns

| Name | Type | Description |
|---|---|---|
| _0 | string | |

## upgradeTo

function upgradeTo(address newImplementation) external nonpayable

*Upgrade the implementation of the proxy to `newImplementation`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address | |

## upgradeToAndCall

function upgradeToAndCall(address newImplementation, bytes data) external payable

*Upgrade the implementation of the proxy to `newImplementation`, and subsequently execute the function call encoded in `data`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address | |
| data | bytes | |

## withdraw

function withdraw(string symbol, uint256 value) external payable

Makes a cross-chain withdraw request to the settlement chain

### Parameters

| Name | Type | Description |
|---|---|---|
| symbol | string | the token symbol to withdraw |

| | | |
|---|---|---|
| value | uint256 | the amount of the withdraw request |

## Events

### AdminChanged

event AdminChanged(address previousAdmin, address newAdmin)

#### Parameters

| Name | Type | Description |
|---|---|---|
| previousAdmin | address | |
| newAdmin | address | |

### BeaconUpgraded

event BeaconUpgraded(address indexed beacon)

#### Parameters

| Name | Type | Description |
|---|---|---|
| beacon `indexed` | address | |

### Initialized

event Initialized(uint8 version)

#### Parameters

| Name | Type | Description |
|---|---|---|
| version | uint8 | |

### Upgraded

event Upgraded(address indexed implementation)

#### Parameters

| Name | Type | Description |
|---|---|---|
| implementation `indexed` | address | |

# FundLock

## Methods

### ALLOWED_WITHDRAWAL_LIMIT

function ALLOWED_WITHDRAWAL_LIMIT() external view returns (uint256)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | uint256 | |

### balanceSheet

function balanceSheet(address userAddress, address tokenAddress) external view returns (uint256)

Getter returning the correct {balanceSheetS} value for the correct token and particular client.

*In cases of `ETH` usage it will convert the `ETH` address to `WETH9` address and read the appropriate balance. See {getIsEtherAndAssetAddress}*

#### Parameters

| Name | Type | Description |
|---|---|---|
| userAddress | address | - address of the user to check balance for |
| tokenAddress | address | - top level token address used (`ETH` for cases with Ether) |

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | uint256 | uint256 - balance amount |

### crossChainDeposit

function crossChainDeposit(address depositor, address tokenAddress, uint256 value) external payable

#### Parameters

| Name | Type | Description |
|---|---|---|
| depositor | address | |
| tokenAddress | address | |
| value | uint256 | |

### crossChainRelease

function crossChainRelease(address withdrawer, string tokenSymbol, address tokenAddress, uint256 withdrawTimestamp, string destinationChain) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| withdrawer | address | |
| tokenSymbol | string | |
| tokenAddress | address | |
| withdrawTimestamp | uint256 | |
| destinationChain | string | |

### crossChainWithdraw

function crossChainWithdraw(address withdrawer, address tokenAddress, uint256 value) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| withdrawer | address | |
| tokenAddress | address | |
| value | uint256 | |

### deposit

function deposit(address tokenAddress, uint256 value) external payable

*User facing function for depositing funds to be used in Ithaca. `value` deposited will go to the client's {balanceSheetS}.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| tokenAddress | address | - address of the token to be deposited |
| value | uint256 | - amount of the token to be deposited |

### executable

function executable() external view returns (contract IFundLockExecutable)

#### Returns

| Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| _0 | contract IFundLockExecutable | |

## fundsToWithdraw

function fundsToWithdraw(address userAddress, address tokenAddress, uint256 index) external view returns (uint256 value, uint256 timestamp)

Getter used to pull data about client's withdraw requests.

*Wraps autogenerated getter for {fundsToWithdrawS} to find a proper token address (in the case of Ether) and read the correct storage slot of the mapping. This will return only ONE request at an index provided.*

**Parameters**

| Name | Type | Description |
|---|---|---|
| userAddress | address | - address of the client who sent withdrawal request(s) |
| tokenAddress | address | - address of the token withdrawn |
| index | uint256 | - index of the particular withdrawal request (max of 5) |

**Returns**

| Name | Type | Description |
|---|---|---|
| value | uint256 | - the amount of funds marked for withdrawal |
| timestamp | uint256 | - timestamp of the exact time withdrawal request was submitted |

## fundsToWithdrawTotal

function fundsToWithdrawTotal(address beneficiary, address token) external view returns (uint256)

Returns the total amount of token flagged for withdrawal for a particular user that still has active {tradeLock}. This is the SUM of all withdraw requests for a particular token and user.

**Parameters**

| Name | Type | Description |
|---|---|---|
| beneficiary | address | - address of the user to get balance for |
| token | address | - address of the token to get balance for |

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | uint256 | uint256 - total amount of client's funds marked for withdrawal |

## gasReceiver

function gasReceiver() external view returns (contract IAxelarGasService)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | contract IAxelarGasService | |

## gateway

function gateway() external view returns (contract IAxelarGateway)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | contract IAxelarGateway | |

## initialize

function initialize(contract IAccessController _accessController, contract ITokenManager _tokenManager, contract IAxelarGateway _gateway, uint256 _tradeLock, uint256 _releaseLock) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| _accessController | contract IAccessController | |
| _tokenManager | contract ITokenManager | |
| _gateway | contract IAxelarGateway | |
| _tradeLock | uint256 | |
| _releaseLock | uint256 | |

## proxiableUUID

function proxiableUUID() external view returns (bytes32)

*Implementation of the ERC1822 {proxiableUUID} function. This returns the storage slot used by the implementation. It is used to validate the implementation's compatibility when performing an upgrade. IMPORTANT: A proxy pointing at a proxiable contract should not be considered proxiable itself, because this risks bricking a proxy that upgrades to it, by delegating to itself until out of gas. Thus it is critical that this function revert if invoked through a proxy. This is guaranteed by the `notDelegated` modifier.*

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bytes32 | |

### registry

function registry() external view returns (contract IRegistry)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | contract IRegistry | |

### release

function release(address tokenAddress, uint256 withdrawTimestamp) external nonpayable

*Actual withdrawal through releasing funds and making transfer to the client who called the function. In order to be released, funds first have to be marked as {fundsToWithdrawS} by the {withdraw()} function and time interval set as {releaseLock} has to pass since withdraw() was called. Uses different transfer logic depending on which token is used (ETH/WETH/ERC20).*

#### Parameters

| Name | Type | Description |
|---|---|---|
| tokenAddress | address | - address of the token to be released |
| withdrawTimestamp | uint256 | - timestamp associated with a certain withdraw request (can be found in {Funds} struct included in the {fundsToWithdrawS} mapping). |

### releaseLock

function releaseLock() external view returns (uint256)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | uint256 | |

### setExecutable

function setExecutable(contract IFundLockExecutable _executable) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| _executable | contract IFundLockExecutable | |

### setGateway

function setGateway(contract IAxelarGateway _gateway) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| _gateway | contract IAxelarGateway | |

## setRegistry

function setRegistry(contract IRegistry _registry) external nonpayable

Setter for {registry} storage var which holds {Registry} address

#### Parameters

| Name | Type | Description |
|---|---|---|
| _registry | contract IRegistry | - {Registry} address to be set |

## setReleaseLockInterval

function setReleaseLockInterval(uint256 interval) external nonpayable

Sets {releaseLock} interval variable on the contract that will be used for all {fundsToWithdraw}. This interval represents time that user has to wait after he called {withdraw()} function before he can {release()} funds. Can only be called by the Admin account.

#### Parameters

| Name | Type | Description |
|---|---|---|
| interval | uint256 | - {releaseLock} interval to be set |

## setTradeLockInterval

function setTradeLockInterval(uint256 interval) external nonpayable

Sets {tradeLock} interval variable on the contract that will be used for all {fundsToWithdraw}. This interval represents time during which user's {fundsToWithdraw} can still be used to cover trades. Starts after {withdraw()} is called, at the same time as {releaseLock}, but is shorter. Can only be called by the Admin account.

#### Parameters

| Name | Type | Description |
|---|---|---|
| interval | uint256 | - {tradeLock} interval to be set |

## settlementChain

function settlementChain() external view returns (string)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | string | |

## settlementContract

function settlementContract() external view returns (string)

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | string | |

### tokenManager

function tokenManager() external view returns (contract ITokenManager)

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | contract ITokenManager | |

### tradeLock

function tradeLock() external view returns (uint256)

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | uint256 | |

### updateBalances

function updateBalances(address[] traders, int256[] amounts, address[] tokens, uint64 backendId) external nonpayable

*Public function used by Ledger to update `balanceSheetS` for all types of accounts. Used by LedgerUpdate flows. Updates trader balances by calling single update function {_updateBalance}, emits event.*

**Parameters**

| Name | Type | Description |
|---|---|---|
| traders | address[] | - array of all clients accounts of the settlement batch |
| amounts | int256[] | - array of amounts of funds to update client balances with |
| tokens | address[] | - array of token addresses for each update row |
| backendId | uint64 | - identificator created by Java Backend to track settlement progress |

### upgradeTo

function upgradeTo(address newImplementation) external nonpayable

*Upgrade the implementation of the proxy to `newImplementation`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address | |

### upgradeToAndCall

function upgradeToAndCall(address newImplementation, bytes data) external payable

*Upgrade the implementation of the proxy to `newImplementation`, and subsequently execute the function call encoded in `data`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address | |
| data | bytes | |

### withdraw

function withdraw(address tokenAddress, uint256 value) external nonpayable

*Marking funds that need to be withdrawn by the depositor. This function does NOT do any transfers. It only marks funds by creating `Funds` structs and mapping them to user and token addresses. Also changes {balanceSheetS} and emits event. Created for security purposes.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| tokenAddress | address | address of the token to be marked for withdrawal |
| value | uint256 | amount of the token to be marked for withdrawal |

## Events

### AdminChanged

event AdminChanged(address previousAdmin, address newAdmin)

#### Parameters

| Name | Type | Description |
|---|---|---|
| previousAdmin | addres | |

| | | |
|---|---|---|
| | s | |
| newAdmin | address | |

## BalanceUpdated

event BalanceUpdated(address indexed user, int256 amount, address indexed token, uint64 indexed backendId)

Event fired upon successful client balance update, which signifies the end of the settlement transaction.

**Parameters**

| Name | Type | Description |
|---|---|---|
| user `indexed` | address | |
| amount | int256 | |
| token `indexed` | address | |
| backendId `indexed` | uint64 | |

## BeaconUpgraded

event BeaconUpgraded(address indexed beacon)

**Parameters**

| Name | Type | Description |
|---|---|---|
| beacon `indexed` | address | |

## FundLockDeposit

event FundLockDeposit(address indexed depositor, address indexed token, uint256 amount)

Event fired after successful deposit to {FundLock}.

**Parameters**

| Name | Type | Description |
|---|---|---|
| depositor `indexed` | address | |
| token `indexed` | address | |
| amount | uint256 | |

### FundsReleased

event FundsReleased(address indexed beneficiary, address indexed token, uint256 amount, uint256 withdrawTimestamp)

Event fired upon releasing funds (tokens) from FundLock to a user's wallet.

#### Parameters

| Name | Type | Description |
|---|---|---|
| beneficiary `indexed` | address | |
| token `indexed` | address | |
| amount | uint256 | |
| withdrawTimestamp | uint256 | |

### FundsToBeWithdrawn

event FundsToBeWithdrawn(address indexed beneficiary, address indexed token, uint256 amount, uint256 totalSlotsUsed)

Event fired with user's {withdraw()} request, which marks funds for release from FundLock.

#### Parameters

| Name | Type | Description |
|---|---|---|
| beneficiary `indexed` | address | |
| token `indexed` | address | |
| amount | uint256 | |
| totalSlotsUsed | uint256 | |

### Initialized

event Initialized(uint8 version)

#### Parameters

| Name | Type | Description |
|---|---|---|
| version | uint8 | |

### RegistryUpdated

event RegistryUpdated(address registry)

#### Parameters

| Name | Type | Description |
|---|---|---|
| registry | address | |

### ReleaseLockSet

event ReleaseLockSet(uint256 interval)

Event fired upod setting the {ReleaseLock} interval.

#### Parameters

| Name | Type | Description |
|---|---|---|
| interval | uint256 | |

### TradeLockSet

event TradeLockSet(uint256 interval)

Event fired upod setting the {TradeLock} interval.

#### Parameters

| Name | Type | Description |
|---|---|---|
| interval | uint256 | |

### Upgraded

event Upgraded(address indexed implementation)

#### Parameters

| Name | Type | Description |
|---|---|---|
| implementation `indexed` | address | |

# Ledger

## Methods

### clientPositions

function clientPositions(uint32, address) external view returns (int64)

*Mapping storing client positions for a specific IThaca market. Map<{uint32 contractId, address clientAddress}, int256 amount>*

#### Parameters

| Name | Type | Description |
|---|---|---|
| _0 | uint32 | |
| _1 | address | |

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | int64 | |

### initialize

function initialize(address _underlyingCurrency, address _strikeCurrency, contract ITokenManager _tokenManager) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| _underlyingCurrency | address | |
| _strikeCurrency | address | |
| _tokenManager | contract ITokenManager | |

### registry

function registry() external view returns (contract IRegistry)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | contract IRegistry | |

### strikeCurrency

function strikeCurrency() external view returns (address)

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | address | |

### tokenManager

function tokenManager() external view returns (contract ITokenManager)

#### Returns

| Name | Type | Descriptio n |
|---|---|---|
| _0 | contract ITokenManager | |

### underlyingCurrency

function underlyingCurrency() external view returns (address)

#### Returns

| Name | Type | Descriptio n |
|---|---|---|
| _0 | addres s | |

### updatePositions

function updatePositions(ILedger.PositionParam[] positions, ILedger.FundMovementParam[] fundMovements, uint64 backendId) external nonpayable

#### Parameters

| Name | Type | Descriptio n |
|---|---|---|
| positions | ILedger.PositionParam[] | |
| fundMovements | ILedger.FundMovementParam[] | |
| backendId | uint64 | |

## Events

### Initialized

event Initialized(uint8 version)

#### Parameters

| Name | Type | Descriptio n |
|---|---|---|
| version | uint8 | |

### LedgerPositionMoved

event LedgerPositionMoved(uint32 indexed contractId, address indexed user, int64 positionSize)

#### Parameters

| Name | Type | Descriptio n |
|---|---|---|
| contractId `indexed` | uint32 | |
| user `indexed` | addres | |

|  |  |  |
|---|---|---|
|  | s |  |
| positionSize | int64 |  |

### PositionsUpdated

event PositionsUpdated(uint64 indexed backendId)

#### Parameters

| Name | Type | Description |
|---|---|---|
| backendId `indexed` | uint64 |  |

# Registry

Registry

*Contract is used to register contracts used in our systems and set/initialize storage of the module. Some methods can be only called by registered contracts. It prevents unwanted external method call by third party client Initialization contract for Registry.*

## Methods

### contractsLength

function contractsLength() external view returns (uint256)

*View that returns amount of registered contracts*

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | uint256 |  |

### deployLedge

function deployLedge(address _underlyingCurrency, address _strikeCurrency, uint256 _precisionUnderlyingCurrency, uint256 _precisionStrikeCurrency) external nonpayable returns (contract LedgerBeaconProxy)

*Deployment of each Ledger market ({Ledger}). First Router contract is deployed, next - Router is added as verified contract. Finally deployment of Ledger contract is executed.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| _underlyingCurrency | address | address of underlyingCurrency asset used in Ledger contract |

| | | |
|---|---|---|
| _strikeCurrency | address | address of strikeCurrency asset used in Ledger contract |
| _precisionUnderlyingCurrency | uint256 | min precision of base token |
| _precisionStrikeCurrency | uint256 | min precision of underlying token |

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | contract LedgerBeaconProxy | |

## fundLock

function fundLock() external view returns (contract IFundLock)

Address of {Fundlock}

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | contract IFundLock | |

## getContractByIdx

function getContractByIdx(uint256 idx) external view returns (address)

*Returning registered contract by idx*

**Parameters**

| Name | Type | Description |
|---|---|---|
| idx | uint256 | address of registered contract |

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | address | |

## initialize

function initialize(contract IAccessController _accessController, contract ITokenManager _tokenManager, contract ITokenValidator _tokenValidator, contract IFundLock _fundLock) external nonpayable

**Parameters**

| Name | Type | Description |
|---|---|---|

|  |  |
|---|---|
| _accessController | contract IAccessController |
| _tokenManager | contract ITokenManager |
| _tokenValidator | contract ITokenValidator |
| _fundLock | contract IFundLock |

## isValidContract

function isValidContract(address _contract) external view returns (bool)

*View that checks if contract is present in Registry storage*

**Parameters**

| Name | Type | Description |
|---|---|---|
| _contract | address | address of contract to validate |

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | bool |  |

## isValidContractOrUtility

function isValidContractOrUtility(address _contract) external view returns (bool)

*Function used by some modifiers or to just check if the a certain contract is registered as a part of IThaca Protocol. Protects from unauthorized calls from external contracts outside of our domain.*

**Parameters**

| Name | Type | Description |
|---|---|---|
| _contract | address | - address of the contract to check |

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | bool |  |

## isValidContractOrUtilityBase

function isValidContractOrUtilityBase(address _contract) external view returns (bool)

*View that checks if contract is present in Registry storage. This provides an access point to a higher level function from Registry.*

**Parameters**

| Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| _contract | address | address of contract to validate |

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bool | |

### ledgerBeacon

function ledgerBeacon() external view returns (contract LedgerBeacon)

Address of {Beacon}

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | contract LedgerBeacon | |

### proxiableUUID

function proxiableUUID() external view returns (bytes32)

*Implementation of the ERC1822 {proxiableUUID} function. This returns the storage slot used by the implementation. It is used to validate the implementation's compatibility when performing an upgrade. IMPORTANT: A proxy pointing at a proxiable contract should not be considered proxiable itself, because this risks bricking a proxy that upgrades to it, by delegating to itself until out of gas. Thus it is critical that this function revert if invoked through a proxy. This is guaranteed by the `notDelegated` modifier.*

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bytes32 | |

### setFundLock

function setFundLock(contract IFundLock _fundLock) external nonpayable

*Setter for {FundLockRouter} address*

#### Parameters

| Name | Type | Description |
|---|---|---|
| _fundLock | contract IFundLock | |

### setLedgerBeacon

function setLedgerBeacon(contract LedgerBeacon _ledgerBeacon) external nonpayable

**Parameters**

| Name | Type | Description |
|---|---|---|
| _ledgerBeacon | contract LedgerBeacon | |

## setTokenManager

function setTokenManager(contract ITokenManager _tokenManager) external nonpayable

*Registers {TokenManager} contract by assigning it's address to the `tokenManager` variable. Only `governor` can call this function.*

**Parameters**

| Name | Type | Description |
|---|---|---|
| _tokenManager | contract ITokenManager | - address of a {TokenManager} contract to be set. |

## setTokenValidator

function setTokenValidator(contract ITokenValidator _tokenValidator) external nonpayable

*Setting {TokenValidator} contract*

**Parameters**

| Name | Type | Description |
|---|---|---|
| _tokenValidator | contract ITokenValidator | address of {TokenValidator} contract to be set in |

## tokenManager

function tokenManager() external view returns (contract ITokenManager)

Address of {TokenManagerRouter}

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | contract ITokenManager | |

## tokenValidator

function tokenValidator() external view returns (contract ITokenValidator)

Address of {TokenValidator}

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | contract ITokenValidator | |

### upgradeTo

function upgradeTo(address newImplementation) external nonpayable

*Upgrade the implementation of the proxy to `newImplementation`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address |  |

### upgradeToAndCall

function upgradeToAndCall(address newImplementation, bytes data) external payable

*Upgrade the implementation of the proxy to `newImplementation`, and subsequently execute the function call encoded in `data`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address |  |
| data | bytes |  |

## Events

### AdminChanged

event AdminChanged(address previousAdmin, address newAdmin)

#### Parameters

| Name | Type | Description |
|---|---|---|
| previousAdmin | address |  |
| newAdmin | address |  |

### BeaconUpgraded

event BeaconUpgraded(address indexed beacon)

#### Parameters

| Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| beacon `indexed` | address | |

### ContractVerified

event ContractVerified(address verifiedContract, uint256 ledgerId)

#### Parameters

| Name | Type | Description |
|---|---|---|
| verifiedContract | address | |
| ledgerId | uint256 | |

### Initialized

event Initialized(uint8 version)

#### Parameters

| Name | Type | Description |
|---|---|---|
| version | uint8 | |

### Upgraded

event Upgraded(address indexed implementation)

#### Parameters

| Name | Type | Description |
|---|---|---|
| implementation `indexed` | address | |

# TokenManager

## Methods

### collectFundsToFundLock

function collectFundsToFundLock(address from, address tokenAddress, uint256 amount) external nonpayable

*Function that allows to safely transfer tokens to {FundLock} from an external source. This function is always called by {FundLock}, so msg.sender is address of {FundLock} contract Can be used only by verified Ledger contracts.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| from | address | address of account to transfer tokens from |
| tokenAddress | address | address of the token to transfer |
| amount | uint256 | of tokens to be transferred |

### ethereumAddress

function ethereumAddress() external view returns (address)

*Address of Ether (ETH). This is used internally to differentiate between ERC20 tokens and ETH. Any fund related operation check token address against this one to verify if the token in the operation is ETH or not. If it is - we need to do a different flow for managing it.*

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | address | |

### initialize

function initialize(contract IAccessController accessController_) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| accessController_ | contract IAccessController | |

### proxiableUUID

function proxiableUUID() external view returns (bytes32)

*Implementation of the ERC1822 {proxiableUUID} function. This returns the storage slot used by the implementation. It is used to validate the implementation's compatibility when performing an upgrade. IMPORTANT: A proxy pointing at a proxiable contract should not be considered proxiable itself, because this risks bricking a proxy that upgrades to it, by delegating to itself until out of gas. Thus it is critical that this function revert if invoked through a proxy. This is guaranteed by the `notDelegated` modifier.*

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | bytes32 | |

### registry

function registry() external view returns (contract IRegistry)

*Address of the {Registry}*

**Returns**

| Name | Type | Description |
|---|---|---|
| _0 | contract IRegistry | |

### setEthereumAddress

function setEthereumAddress(address _ethereumAddress) external nonpayable

*Setting ethereum asset address in {TokenManager}*

**Parameters**

| Name | Type | Description |
|---|---|---|
| _ethereumAddress | address | address of ethereum asset |

### setRegistry

function setRegistry(contract IRegistry _registry) external nonpayable

Setter for {registry} storage var which holds {RegistryRouter} address

**Parameters**

| Name | Type | Description |
|---|---|---|
| _registry | contract IRegistry | - {RegistryRouter} address to be set |

### setWETH9Address

function setWETH9Address(address _weth9Address) external nonpayable

*Setting {WETH9} asset address in {TokenManager}*

**Parameters**

| Name | Type | Description |
|---|---|---|
| _weth9Address | address | address of {WETH9} contract |

### upgradeTo

function upgradeTo(address newImplementation) external nonpayable

*Upgrade the implementation of the proxy to `newImplementation`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

**Parameters**

| Name | Type | Description |
|---|---|---|
| newImplementation | addres | |

s

### upgradeToAndCall

function upgradeToAndCall(address newImplementation, bytes data) external payable

*Upgrade the implementation of the proxy to `newImplementation`, and subsequently execute the function call encoded in `data`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| newImplementation | address | |
| data | bytes | |

### weth9Address

function weth9Address() external view returns (address)

*Address of Wrapped Ethereum (WETH). This is used internally to differentiate between other ERC20 tokens and WETH. Any fund related operation check token address against this one to verify if the token in the operation is WETH or not. If it is - we need to do a different flow for managing it.*

#### Returns

| Name | Type | Description |
|---|---|---|
| _0 | address | |

## Events

### AdminChanged

event AdminChanged(address previousAdmin, address newAdmin)

#### Parameters

| Name | Type | Description |
|---|---|---|
| previousAdmin | address | |
| newAdmin | address | |

### BeaconUpgraded

event BeaconUpgraded(address indexed beacon)

#### Parameters

| Name | Type | Description |
|---|---|---|
| beacon `indexed` | address | |

### EthereumAddressSet

event EthereumAddressSet(address etherAddress)

#### Parameters

| Name | Type | Description |
|---|---|---|
| etherAddress | address | |

### IThacaRegistrySet

event IThacaRegistrySet(address IThacaRegistryAddress)

#### Parameters

| Name | Type | Description |
|---|---|---|
| IThacaRegistryAddress | address | |

### Initialized

event Initialized(uint8 version)

#### Parameters

| Name | Type | Description |
|---|---|---|
| version | uint8 | |

### Upgraded

event Upgraded(address indexed implementation)

#### Parameters

| Name | Type | Description |
|---|---|---|
| implementation `indexed` | address | |

### WETH9AddressSet

event WETH9AddressSet(address weth9Address)

#### Parameters

| Name | Type | Description |
|---|---|---|
| weth9Address | address | |

# TokenValidator

TokenValidator

*Contract used for whitelisting tokens used in IThaca Markets Only user with "admin" role can add or remove tokens to whitelist If token is not valid, transaction is reverted Additionally contract is storing precision values for tokens. Precision is used in trade settlement calculations*

## Methods

### addTokensToWhitelist

function addTokensToWhitelist(ITokenValidator.AddTokensToWhitelistParams[] params) external nonpayable

#### Parameters

| Name | Type | Description |
|---|---|---|
| params | ITokenValidator.AddTokensToWhitelistParams[] | |

### getTokenPrecision

function getTokenPrecision(address token) external view returns (uint256 precision, uint256 toTokenPower)

*Function to get precision for token.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| token | address | address of token to get precision for |

#### Returns

| Name | Type | Description |
|---|---|---|
| precision | uint256 | of a requested token |
| toTokenPower | uint256 | to which token should be raised in IThaca calculations |

### initialize

function initialize(contract IAccessController accessController_) external nonpayable

#### Parameters

| Name | Type | Descriptio n |
|---|---|---|
| accessController _ | contract IAccessController | |

### proxiableUUID

function proxiableUUID() external view returns (bytes32)

*Implementation of the ERC1822 {proxiableUUID} function. This returns the storage slot used by the implementation. It is used to validate the implementation's compatibility when performing an upgrade. IMPORTANT: A proxy pointing at a proxiable contract should not be considered proxiable itself, because this risks bricking a proxy that upgrades to it, by delegating to itself until out of gas. Thus it is critical that this function revert if invoked through a proxy. This is guaranteed by the `notDelegated` modifier.*

#### Returns

| Name | Type | Descriptio n |
|---|---|---|
| _0 | bytes3 2 | |

### removeTokenFromWhitelist

function removeTokenFromWhitelist(address token) external nonpayable

*Function to remove token address from {TokenValidator} whitelist*

#### Parameters

| Name | Type | Description |
|---|---|---|
| token | address | address of token to be removed from {TokenValidator} whitelist |

### upgradeTo

function upgradeTo(address newImplementation) external nonpayable

*Upgrade the implementation of the proxy to `newImplementation`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

#### Parameters

| Name | Type | Descriptio n |
|---|---|---|
| newImplementation | addres s | |

### upgradeToAndCall

function upgradeToAndCall(address newImplementation, bytes data) external payable

*Upgrade the implementation of the proxy to `newImplementation`, and subsequently execute the function call encoded in `data`. Calls {_authorizeUpgrade}. Emits an {Upgraded} event.*

**Parameters**

| Name | Type | Description |
|---|---|---|
| newImplementation | address | |
| data | bytes | |

### validateToken

function validateToken(address token) external view

*Function to check if token can be used in IThaca as deposit in {FundLock} and traded with*

**Parameters**

| Name | Type | Description |
|---|---|---|
| token | address | address of token to be checked |

### validateTokens

function validateTokens(address[] tokens) external view

*Function to bulk check if tokens can be used in IThaca as deposit in {FundLock} and traded with*

**Parameters**

| Name | Type | Description |
|---|---|---|
| tokens | address[] | addresses of tokens to be checked |

### whitelistedTokens

function whitelistedTokens(address) external view returns (uint256 precision, uint256 toTokenPower)

Mapping holding datd for IThaca whitelisted token precisions.

*Maps token address to a {Precision} struct holding precision values. Notice that a zero precision can NOT be passed since it serves as a sign that a precision for a certain token has not been initialized. See {Precision}*

**Parameters**

| Name | Type | Description |
|---|---|---|
| _0 | address | |

**Returns**

| Name | Type | Description |
|---|---|---|
| precision | uint256 | |

| | |
|---|---|
| toTokenPower | uint256 |

## Events

### AdminChanged

event AdminChanged(address previousAdmin, address newAdmin)

#### Parameters

| Name | Type | Description |
|---|---|---|
| previousAdmin | address | |
| newAdmin | address | |

### BeaconUpgraded

event BeaconUpgraded(address indexed beacon)

#### Parameters

| Name | Type | Description |
|---|---|---|
| beacon `indexed` | address | |

### Initialized

event Initialized(uint8 version)

#### Parameters

| Name | Type | Description |
|---|---|---|
| version | uint8 | |

### TokenRemovedFromWhitelist

event TokenRemovedFromWhitelist(address indexed deletedToken)

*Event fired upon token delisting.*

#### Parameters

| Name | Type | Description |
|---|---|---|
| deletedToken `indexed` | address | - address of a token removed from whitelist |

### TokenWhitelisted

event TokenWhitelisted(address indexed whitelistedToken, uint256 precision, uint256 toTokenPower)

*Event fired upon whitelisting a token for IThaca usage.*

**Parameters**

| Name | Type | Description |
|---|---|---|
| whitelistedToken `indexed` | address | - address of a whitelisted token |
| precision | uint256 | - amount of decimals used/important for IThaca during rounding of values (always <= token decimals!) |
| toTokenPower | uint256 | - difference between decimals and precision, this value is used for calculations to convert (normalize) precision based amount to actual token denomination 10^(decimals-precision) |

## Upgraded

event Upgraded(address indexed implementation)

**Parameters**

| Name | Type | Description |
|---|---|---|
| implementation `indexed` | address | |