MUSD - Technical Specification

1. Overview

MUSD is an upgradeable ERC20-compliant token contract that extends the functionality of a base yield-bearing token (**MYieldToOne**). It introduces features for pausing contract operations and allowing privileged roles to seize assets from frozen accounts.

The contract inherits from:

- `IMUSD`: An interface defining MUSD-specific events and roles.
- `MYieldToOne`: A base contract providing yield-bearing token mechanics where yield is directed to a single recipient. It also includes `Freezable` functionality.
- `PausableUpgradeable`: OpenZeppelin's standard pausable module for upgradeable contracts.

2. Contract Details

- **Path:** `src/MUSD.sol`
- **Solidity Version:** `0.8.26`
- **License:** `BUSL-1.1`

3. Roles and Access Control

The contract uses an `AccessControl` mechanism with the following roles:

Role	Hash	Description
`DEFAULT_ADMIN_ROLE`	`0x00`	The super-user role that can grant and revoke other roles.
`FREEZE_MANAGER_ROLE`	`keccak256("FREEZE_MANAGER_ROLE")`	Can freeze and unfreeze accounts.
`YIELD_RECIPIENT_MANAGER_ROLE`	`keccak256("YIELD_RECIPIENT_MANAGER_ROLE")`	Can change the yield recipient address and claim yield.

Role	Hash	Description
`PAUSER_ROLE`	`keccak256("PAUSER_ROLE")`	Can pause and unpause the contract, halting most token operations.
`FORCED_TRANSFER_MANAGER_ROLE`	`keccak256("FORCED_TRANSFER_MANAGER_ROLE")`	Can seize funds from a frozen account and transfer them to another address.

4. State Variables

The contract defines the following public constants for role identification:

Name	Type	Visibility	Description
`PAUSER_ROLE`	`bytes32`	`public`	The identifier for the pauser role.
`FORCED_TRANSFER_MANAGER_ROLE`	`bytes32`	`public`	The identifier for the forced transfer manager role.

5. Functions

5.1. Constructor

The constructor is designed for the implementation contract and is not called when interacting with the proxy. It constructs the parent `MYieldToOne` contract.

```
constructor(address mToken, address swapFacility) MYieldToOne(mToken, swapFacility)
```

- **Parameters:**
  - `mToken`: The address of the `M` token contract.
  - `swapFacility`: The address of the `SwapFacility` contract.
- **Logic:**

1. Calls the `MYieldToOne` constructor, passing along the `mToken` and `swapFacility` addresses.
2. The `MExtension` constructor (called by `MYieldToOne`) disables initializers for the implementation contract by calling `_disableInitializers()`.

5.2. Initializer

initialize

Initializes the MUSD contract state. This function is called once when the proxy is deployed.

```
function initialize(
    address yieldRecipient,
    address admin,
    address freezeManager,
    address yieldRecipientManager,
    address pauser,
    address forcedTransferManager
) public virtual initializer
```

- **Parameters:**
  - `yieldRecipient`: The address that will receive the generated yield.
  - `admin`: The address to be granted the `DEFAULT_ADMIN_ROLE`.
  - `freezeManager`: The address to be granted the `FREEZE_MANAGER_ROLE`.
  - `yieldRecipientManager`: The address to be granted the `YIELD_RECIPIENT_MANAGER_ROLE`.
  - `pauser`: The address to be granted the `PAUSER_ROLE`.
  - `forcedTransferManager`: The address to be granted the `FORCED_TRANSFER_MANAGER_ROLE`.
- **Logic:**
  1. Initializes the parent `MYieldToOne` contract with token details, admin, and managers.
  2. Initializes the `PausableUpgradeable` state.
  3. Grants `PAUSER_ROLE` and `FORCED_TRANSFER_MANAGER_ROLE` to the specified addresses.
- **Errors:**
  - `ZeroPauser()`: If `pauser` is `address(0)`.

- `ZeroForcedTransferManager()`: If `forcedTransferManager` is `address(0)`.

5.3. Pausable Functions

`pause`

Pauses the contract. When paused, `approve`, `wrap` , `unwrap` and `transfer` are disabled.

```
function pause() public onlyRole(PAUSER_ROLE)
```

- **Modifiers:** `onlyRole(PAUSER_ROLE)`
- **Events:** `Paused(address account)`

`unpause`

Unpauses the contract, restoring normal operations.

```
function unpause() public onlyRole(PAUSER_ROLE)
```

- **Modifiers:** `onlyRole(PAUSER_ROLE)`
- **Events:** `Unpaused(address account)`

5.4. Forced Transfer Functions

`forceTransfer`

Allows an address with the `FORCED_TRANSFER_MANAGER_ROLE` to seize a specific `amount` of tokens from a `frozenAccount` and send them to a `recipient`. This function bypasses pause and freeze checks.

```
function forceTransfer(
    address frozenAccount,
    address recipient,
    uint256 amount
) external onlyRole(FORCED_TRANSFER_MANAGER_ROLE)
```

- **Modifiers:** `onlyRole(FORCED_TRANSFER_MANAGER_ROLE)`
- **Events:**
  - `Transfer(address from, address to, uint256 amount)`
  - `ForcedTransfer(address indexed from, address indexed to, address indexed enforcer, uint256 amount)`

- **Errors:**
  - `InvalidRecipient(address recipient)`: If `recipient` is the zero address.
  - `NotFrozen(address account)`: If `frozenAccount` is not currently frozen.
  - `InsufficientBalance(address account, uint256 balance, uint256 amount)`: If `frozenAccount` has an insufficient balance.
- **Return**:
  - returns early if `amount` is `0`

forceTransfers

A batch version of `forceTransfer` to execute multiple seizures in a single transaction.

```
function forceTransfers(
    address[] calldata frozenAccounts,
    address[] calldata recipients,
    uint256[] calldata amounts
) external onlyRole(FORCED_TRANSFER_MANAGER_ROLE)
```

- **Modifiers:** `onlyRole(FORCED_TRANSFER_MANAGER_ROLE)`
- **Errors:**
  - `ArrayLengthMismatch()`: If the input array lengths are not equal.
  - Inherits all errors from `forceTransfer` and revert the entire batch if one error occurs

6. Hooks and Overrides

`MUSD` overrides several internal functions (hooks) from its parent contracts to inject custom logic.

_beforeApprove

Hook called before an `approve` operation.

- **Logic:** Reverts if the contract is paused.

_beforeWrap

Hook called before a `wrap` (mint) operation.

- **Logic:** Reverts if the contract is paused.

_beforeUnwrap

Hook called before an `unwrap` (burn) operation.

- **Logic:** Reverts if the contract is paused.

`_beforeTransfer`

Hook called before a `transfer` or `transferFrom` operation.

- **Logic:** Reverts if the contract is paused.

`_beforeClaimYield`

Hook called before yield is claimed.

- **Logic:** Restricts the caller to only be the `YIELD_RECIPIENT_MANAGER_ROLE`.

7. Events

`ForcedTransfer`

Emitted when tokens are seized from a frozen account.

```
event ForcedTransfer(address indexed from, address indexed to, address indexed enf
orcer, uint256 amount);
```

8. Errors

`ZeroPauser`

Reverts during initialization if the pauser address is the zero address.

```
error ZeroPauser();
```

`ZeroForcedTransferManager`

Reverts during initialization if the forced transfer manager address is the zero address.

```
error ZeroForcedTransferManager();
```

`ArrayLengthMismatch`

Reverts in `forceTransfers` if the input arrays do not have the same length.

```
error ArrayLengthMismatch();
```

MYieldToOne - Technical Specification

1. Overview

`MYieldToOne.sol` is an abstract, upgradeable ERC20-compliant contract that serves as a base for creating non-rebasing tokens that wrap the `M` token. The core feature of this contract is that all yield generated by the underlying `M` tokens is directed to a single, configurable `yieldRecipient`. It integrates `Freezable` functionality to allow freezing of individual accounts, preventing them from participating in token operations.

The contract inherits from:

- `IMYieldToOne`: An interface defining events and functions specific to this yield mechanism.
- `MYieldToOneStorageLayout`: An abstract contract defining the storage layout to prevent clashes in inheriting contracts.
- `MExtension`: A base contract providing core ERC20 logic and interaction with the `M` token and `SwapFacility`.
- `Freezable`: A component providing account freezing and unfreezing capabilities.

2. Contract Details

- **Path:** `lib/evm-m-extensions/src/projects/yieldToOne/MYieldToOne.sol`
- **Solidity Version:** `0.8.26`
- **License:** `BUSL-1.1`

3. Storage Layout

The contract uses a dedicated storage struct `MYieldToOneStorageStruct` located at a specific storage slot to prevent collisions.

```
struct MYieldToOneStorageStruct {
    uint256 totalSupply;
    address yieldRecipient;
    mapping(address account ⇒ uint256 balance) balanceOf;
}
```

4. Roles and Access Control

Role	Hash	Description
`DEFAULT_ADMIN_ROLE`	`0x00`	Can grant and revoke other roles.
`FREEZE_MANAGER_ROLE`	`keccak256("FREEZE_MANAGER_ROLE")`	Can freeze and unfreeze accounts.
`YIELD_RECIPIENT_MANAGER_ROLE`	`keccak256("YIELD_RECIPIENT_MANAGER_ROLE")`	Can change the yield recipient address and claim yield.

5. Functions

5.1. Constructor

The constructor constructs the `MExtension` parent contract, setting immutable addresses for the `M` token and `SwapFacility`. This constructor is intended for the implementation contract and is not called on the proxy.

```
constructor(address mToken, address swapFacility) MExtension(mToken, swapFacilit
y)
```

- **Parameters:**
  - `mToken`: The address of the `M` token contract.
  - `swapFacility`: The address of the `SwapFacility` contract.
- **Logic:**
  1. Calls the `MExtension` constructor to store the immutable `mToken` and `swapFacility` addresses.
  2. The `MExtension` constructor also calls `_disableInitializers()` to prevent the implementation contract from being initialized.

5.2. Initializer

`initialize` / `__MYieldToOne_init`

Initializes the contract state.

```
function initialize(
    string memory name,
```

```
    string memory symbol,
    address yieldRecipient_,
    address admin,
    address freezeManager,
    address yieldRecipientManager
) public virtual initializer
```

- **Logic:**
  1. Initializes parent contracts `MExtension` and `Freezable`.
  2. Sets the initial `yieldRecipient`.
  3. Grants `DEFAULT_ADMIN_ROLE` and `YIELD_RECIPIENT_MANAGER_ROLE`.
- **Errors:**
  - `ZeroYieldRecipientManager()`: If `yieldRecipientManager` is `address(0)`.
  - `ZeroAdmin()`: If `admin` is `address(0)`.
  - `ZeroYieldRecipient()`: If `yieldRecipient_` is `address(0)`.

5.3. Interactive Functions

claimYield

Claims the accumulated yield and transfers it to the `yieldRecipient`. The yield is calculated as the difference between the contract's `M` token balance and its `totalSupply`.

```
function claimYield() public returns (uint256)
```

- **Events:** `YieldClaimed(uint256 amount)`
- **Logic:**
  1. Calls the `_beforeClaimYield` hook.
  2. Calculates the available `yield`.
  3. If yield is greater than zero, mints new tokens equivalent to the yield amount to the `yieldRecipient`.
- **Return**:
  - returns early if `yield` equals `0`

setYieldRecipient

Updates the address of the yield recipient. Before setting the new recipient, it claims any pending yield for the old one.

```
function setYieldRecipient(address account) external onlyRole(YIELD_RECIPIENT_MANAGER_ROLE)j
```

- **Modifiers:** `onlyRole(YIELD_RECIPIENT_MANAGER_ROLE)`
- **Events:** `YieldRecipientSet(address indexed recipient)`
- **Errors:** `ZeroYieldRecipient()`: If `account` is `address(0)`.

5.3. View Functions

`balanceOf`

Returns the token balance of an account.

`totalSupply`

Returns the total supply of the token.

`yield`

Calculates the currently available, unclaimed yield.

`yieldRecipient`

Returns the current yield recipient's address.

6. Hooks

`MYieldToOne` overrides hooks from `MExtension` and `Freezable` to enforce freeze checks. It also introduces a new hook.

- `_beforeApprove`: Reverts if the `account` or `spender` is frozen.
- `_beforeWrap`: Reverts if the `account` or `recipient` is frozen.
- `_beforeUnwrap`: Reverts if the `account` is frozen.
- `_beforeTransfer`: Reverts if the `msg.sender`, `sender`, or `recipient` is frozen.
- `_beforeClaimYield`: An empty virtual hook designed to be overridden by child contracts for adding pre-claim logic.

7. Internal Functions

- `_mint`: Mints tokens and increases `totalSupply`.
- `_burn`: Burns tokens and decreases `totalSupply`.
- `_update`: Updates balances for transfers.
- `_setYieldRecipient`: Internal logic to set the yield recipient.

8. Events

- `YieldClaimed(uint256 amount)`: Emitted when yield is claimed.
- `YieldRecipientSet(address indexed recipient)`: Emitted when the yield recipient is changed.

9. Errors

- `ZeroYieldRecipientManager`: Reverts if the yield recipient manager address is zero during initialization.
- `ZeroAdmin`: Reverts if the admin address is zero during initialization.
- `ZeroYieldRecipient`: Reverts if the yield recipient address is set to zero.

MExtension - Technical Specification

1. Overview

`MExtension.sol` is an abstract, upgradeable ERC20 contract that forms the foundational layer for wrapping the `M` token into various non-rebasing token extensions. It handles the core logic for interacting with the `M` token and the `SwapFacility`, which is the sole entry point for `wrap` and `unwrap` operations. The contract is designed to be extended, providing virtual hooks and abstract functions that must be implemented by child contracts to define their specific token mechanics (e.g., balance storage, transfer logic).

The contract inherits from:

- `IMExtension`: An interface defining the core functions for an M-token extension.
- `ERC20ExtendedUpgradeable`: An extended version of OpenZeppelin's ERC20 standard for upgradeable contracts.

2. Contract Details

- **Path:** `lib/evm-m-extensions/src/MExtension.sol`
- **Solidity Version:** `0.8.26`

- **License:** `BUSL-1.1`

3. State Variables

Name	Type	Visibility	Mutability	Description
`mToken`	`address`	`public`	`immutable`	The address of the `M` token contract.
`swapFacility`	`address`	`public`	`immutable`	The address of the `SwapFacility` contract.

4. Functions

4.1. Constructor

Initializes the immutable state variables of the contract. This is intended for the implementation contract and is not called on the proxy.

```
constructor(address mToken_, address swapFacility_)
```

- **Logic:**
  1. Disables initializers to prevent re-initialization of the implementation contract.
  2. Sets the `mToken` and `swapFacility` addresses.
- **Errors:**
  - `ZeroMToken()`: If `mToken_` is `address(0)`.
  - `ZeroSwapFacility()`: If `swapFacility_` is `address(0)`.

4.2. Initializer

`__MExtension_init`

Initializes the ERC20 properties of the extension token.

```
function __MExtension_init(string memory name, string memory symbol) internal onlyInitializing
```

- **Logic:** Calls the `__ERC20ExtendedUpgradeable_init` initializer to set the token's name, symbol, and decimals (hardcoded to 6).

4.3. Interactive Functions

wrap

Wraps `M` tokens into the extension token. This can only be called by the `SwapFacility`.

```
function wrap(address recipient, uint256 amount) external onlySwapFacility
```

unwrap

Unwraps the extension token back into `M` tokens. This can only be called by the `SwapFacility`.

```
function unwrap(address recipient, uint256 amount) external onlySwapFacility
```

enableEarning / disableEarning

Controls whether the contract's `M` token balance earns yield.

- **Events:**
  - `EarningEnabled(uint128 index)`
  - `EarningDisabled(uint128 index)`
- **Errors:**
  - `EarningIsEnabled()`: If earning is already enabled.
  - `EarningIsDisabled()`: If earning is already disabled.

4.4. View Functions

- **`currentIndex()`**: Returns the `M` token's current yield index.
- **`isEarningEnabled()`**: Returns `true` if the contract is earning `M` token's yield.
- **`balanceOf(address account)`**: Abstract function to be implemented by child contracts.

5. Hooks

`MExtension` provides empty virtual hooks to allow child contracts to inject logic at different stages of the token lifecycle.

- `_beforeApprove`
- `_beforeWrap`
- `_beforeUnwrap`
- `_beforeTransfer`

6. Internal Functions

6.1. Core Logic

`_approve`

Overrides the standard `_approve` function to inject custom logic via the `_beforeApprove` hook before calling the parent implementation.

`_wrap`

Internal logic for wrapping `M` into the extension token. It performs necessary checks, calls the `_beforeWrap` hook, transfers `M` tokens from the `SwapFacility`, and then calls the abstract `_mint` function, which must be implemented by the child contract.

`_unwrap`

Internal logic for unwrapping the extension token back to `M`. It performs checks, calls the `_beforeUnwrap` hook, calls the abstract `_burn` function, and transfers the `M` tokens back to the `SwapFacility`.

`_transfer`

Overrides the standard `_transfer` function. It performs checks for invalid recipients and insufficient balances, calls the `_beforeTransfer` hook, emits the `Transfer` event, and finally calls the abstract `_update` function to modify token balances.

6.2. Abstract Functions

These functions must be implemented by any contract that inherits from `MExtension`.

- **`_mint(address recipient, uint256 amount)`**: Defines how tokens are created.
- **`_burn(address account, uint256 amount)`**: Defines how tokens are destroyed.
- **`_update(address sender, address recipient, uint256 amount)`**: Defines the logic for updating balances during a transfer.

7. Events

- **`EarningEnabled(uint128 index)`**: Emitted when the contract starts earning yield.
- **`EarningDisabled(uint128 index)`**: Emitted when the contract stops earning yield.

8. Errors

- **`NotSwapFacility()`**: Caller is not the `swapFacility`.

- `ZeroMToken()`: `mToken` address is zero.
- `ZeroSwapFacility()`: `swapFacility` address is zero.
- `EarningIsEnabled()`: Earning is already enabled.
- `EarningIsDisabled()`: Earning is already disabled.
- `InvalidRecipient(address recipient)`: Recipient is the zero address.
- `InsufficientAmount(uint256 amount)`: Token amount is zero.
- `InsufficientBalance(address account, uint256 balance, uint256 amount)`: Account has an insufficient balance.