Contract Manager

Overview

ContractManager is a smart contract designed to manage a set of contract addresses along with their associated descriptions. This contract allows for the addition, updating, and removal of addresses while leveraging access control mechanisms to ensure that only authorised users can perform these actions.

Data Structures

mapping(address => string) public addDescription

1. Description: A mapping from contract addresses to their descriptions.
2. Usage: Stores the description associated with each address.
3. Reason to use this data structure: Simplicity was the motto, as simpler code is always less prone to errors and bugs.
4. Using a mapping provides direct access to the description of a contract address. This is particularly useful in scenarios where descriptions need to be quickly retrieved or updated without having to iterate over a list of addresses.
5. Mappings in Solidity allow for constant-time complexity (O(1)) lookups.

Functions

1. **function addAddresses(address newAddress, string calldata description) external;**
    a. Description: Adds a new address to the mapping with a given description.
    b. Arguments:
        i. newAddress: The address to be added.
        ii. description: The description of the address.
    c. Access Control: Only users with the appropriate permissions can call this function.
    d. Reverts:
        i. If the address is zero.
        ii. If the address already exists.
    e. Events:
        i. Emits AddAddress on successful addition of the address.

2. **function updateDescription(address existingAddress, string calldata description) external;**
    a. Description: Updates the description of an existing address in the mapping.
    b. Arguments:
        i. existingAddress: The existing address whose description is to be updated.
        ii. description: Updated description.
    c. Access Control: Only users with the appropriate permissions can call this function.
    d. Reverts:
        i. If the address does not exist.

e. Events:
    i. Emits UpdateDescription on successful update of the description.

3. **function removeAddress(address existingAddress) external;**
    a. Description: Removes an existing address and its associated description from the mapping.
    b. Arguments:
        i. existingAddress: Existing address that has to be removed.
    c. Access Control: Only users with the appropriate permissions can call this function.
    d. Reverts:
        i. If the address does not exist.
    e. Events:
        i. Emits RemoveAddress on successful removal of the address.

4. **function _checkAccessAllowed(string memory signature) internal view;**
    a. Description: Internal function to verify if the caller has permission to call a specific function. Reverts with an Unauthorised error if not allowed.
    b. Arguments:
        i. signature: The function signature to check
    c. Access Control: Used internally to enforce access control.

5. **function _isValidContractAddress(address contractAddress) internal view returns (bool);**
    a. Description: This function checks whether the provided address is a smart contract address.
    b. Arguments:
        i. contractAddress: The address to be validated as a contract address.
    c. Return Value:
        i. Returns true if the contractAddress has associated code (i.e., it is a smart contract address).
        ii. Returns false if the contractAddress does not have associated code (i.e., it is an externally owned account (EOA) or an invalid address).

Access Control Mechanisms

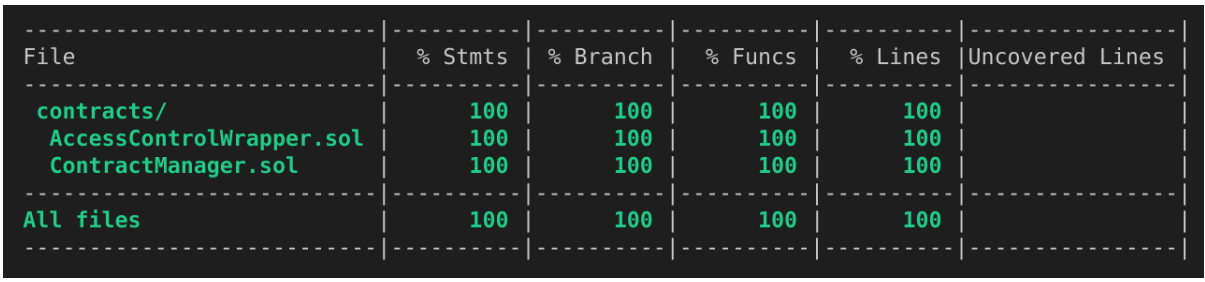
Inherited from **AccessControlWrapper**

1. Role-Based Access Control: Utilises roles to manage permissions for different functions.
2. Functions:
    a. giveCallPermission: Grants permission to an account for a specific contract function.
    b. revokeCallPermission: Revokes permission from an account for a specific contract function.

c. isAllowedToCall: Checks if an account has permission to call a specific function.

Testing Approach

1. Unit Tests:
    - Functionality Tests: Ensure that addAddresses, updateDescription, and removeAddress functions work as expected.
    - Access Control Tests: Verify that only authorised users can call the restricted functions.
    - Reversion Tests: Check that the functions revert with the correct error messages under invalid conditions.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
AccessControlWrapper.sol	100	100	100	100	
ContractManager.sol	100	100	100	100	
All files	100	100	100	100	

2. On chain Tests:
    - End-to-End Tests: Simulate real-world scenarios to verify that the contract works correctly in a deployed environment.
    - Event Emission Tests: Confirm that the appropriate events are emitted during function execution.

Example Test Cases

1. Adding an Address:
    - Test adding a valid address and description.
    - Ensure the function reverts when adding a zero address or an already existing address.
    - Verify that the AddAddress event is emitted.

    https://sepolia.etherscan.io/tx/0xe47ed059ee072b6c9cdfd05ba8e6221dfcb646872960a627e78b36fcfcce2393

2. Updating an Address:
    - Test updating the description of an existing address.
    - Ensure the function reverts when updating a non-existent address.
    - Verify that the UpdateDescription event is emitted.

    https://sepolia.etherscan.io/tx/0x7f06861eccf3a423487c0c33161534bcbb13f1a512671152829807230f5c8679

3. Removing an Address:
    - Test removing an existing address.
    - Ensure the function reverts when removing a non-existent address.

- Verify that the RemoveAddress event is emitted.

https://sepolia.etherscan.io/tx/0x4918fed84ee1129f7043546896d1cd56744f0a742b4db113fa9aaff875f7865d

Security & Efficiency consideration

1. Address that is to be added must be a valid contract address.
    a. _isValidContractAddress() functions checks the validity of the functions.
    b. This check is implemented only in the addAddresses() function.
    c. This also ensures zero address as invalid address.
2. Avoid unauthorised access
    a. AccessControlWrapper contract ensures this.
    b. There are access restrictions in all the external functions.
3. Used custom Errors to save gas.
4. Avoid complex data structures to make code more efficient.
5. Avoid external calls for better security and efficiency.
6. This contract is designed with a focus on security by making it non-upgradeable.
7. Events will be emitted after each interaction or function execution.
8. Make necessary functions view, which are not modifying states.

Assumptions

1. There will be no empty description in the addDescription mapping corresponding to any contract.
2. Contract is non-upgradeable.
3. Revert if the same address tries to be added again.
4. While updating an address, the given address must be present in the addDescription mapping.