# Contract Manager Smart Contract

### **Core Components**

## **Enum: AuthorizationLevel**

*enum AuthorizationLevel { BasicAuth, CanAdd, CanEdit, CanDelete }*

Defines the levels of authorization for users:

* BasicAuth: Default level with no special permissions.
* CanAdd: Permission to add new contract addresses.
* CanEdit: Permission to update contract descriptions.
* CanDelete: Permission to remove contract addresses.

## **Struct: Authorizedusers**

struct Authorizedusers {  
 address useraddress;  
 AuthorizationLevel authorizelevel;  
}

Holds information about an authorized user:

* useraddress: The Ethereum address of the user.
* authorizelevel: The authorization level assigned to the user.

## **State Variables**

* contractDescriptions: A mapping from contract addresses to their descriptions.
* contractAddresses: An array of all contract addresses stored in the registry.
* authorizedusers: A mapping from user addresses to their authorization details.
* owner: The address of the contract owner, who has special permissions.

## **Events**

* ContractAdded(address indexed contractAddress, string description): Emitted when a new contract address is added with a description.
* ContractUpdated(address indexed contractAddress, string newDescription): Emitted when an existing contract address's description is updated.
* ContractRemoved(address indexed contractAddress): Emitted when a contract address is removed from the registry.

## **Modifiers**

* onlyOwner: Restricts access to functions so that only the contract owner can call them.
* canAdd: Allows access to functions for the owner or users with CanAdd permission.
* canUpdate: Allows access to functions for the owner or users with CanEdit permission.
* canDelete: Allows access to functions for the owner or users with CanDelete permission.

## **Constructor**

constructor()

Initializes the contract and sets the deployer as the owner.

## **Functions**

### **adduserrole(address useraddress, AuthorizationLevel \_authorizelevel)**

* **Description**: Assigns an authorization level to a user address.
* **Access**: onlyOwner
* **Parameters**:
  + useraddress: The address of the user.
  + \_authorizelevel: The authorization level to assign.

### **addContract(address \_contractAddress, string memory \_description)**

* **Description**: Adds a new contract address and its description to the registry.
* **Access**: canAdd
* **Parameters**:
  + \_contractAddress: The address of the contract.
  + \_description: A description for the contract.
* **Requires**:
  + Description cannot be empty.
  + Contract address must not already exist.

### **updateDescription(address \_contractAddress, string memory \_newDescription)**

* **Description**: Updates the description of an existing contract address.
* **Access**: canUpdate
* **Parameters**:
  + \_contractAddress: The address of the contract.
  + \_newDescription: The new description for the contract.
* **Requires**:
  + Description cannot be empty.
  + Contract address must exist.

### **removeContract(address \_contractAddress)**

* **Description**: Removes a contract address and its description from the registry.
* **Access**: canDelete
* **Parameters**:
  + \_contractAddress: The address of the contract to remove.
* **Requires**:
  + Contract address must exist.

### **getDescription(address \_contractAddress)**

* **Description**: Retrieves the description of a given contract address.
* **Access**: Public
* **Parameters**:
  + \_contractAddress: The address of the contract.
* **Returns**: The description of the contract.

### **getAllContracts()**

* **Description**: Returns an array of all contract addresses in the registry.
* **Access**: Public
* **Returns**: An array of contract addresses.

### **Summary**

The ContractManager smart contract is designed with a focus on security, efficiency, and clarity. The use of mappings for storage, arrays for iteration, and role-based access control ensures that the contract can manage contract records effectively while safeguarding against unauthorized access. The modular design of functions and the use of events for critical actions enhance the contract’s usability and integration with external systems.