Identity DAO Spec v0.3

dOrg – GoodDollar Project

06.18.19

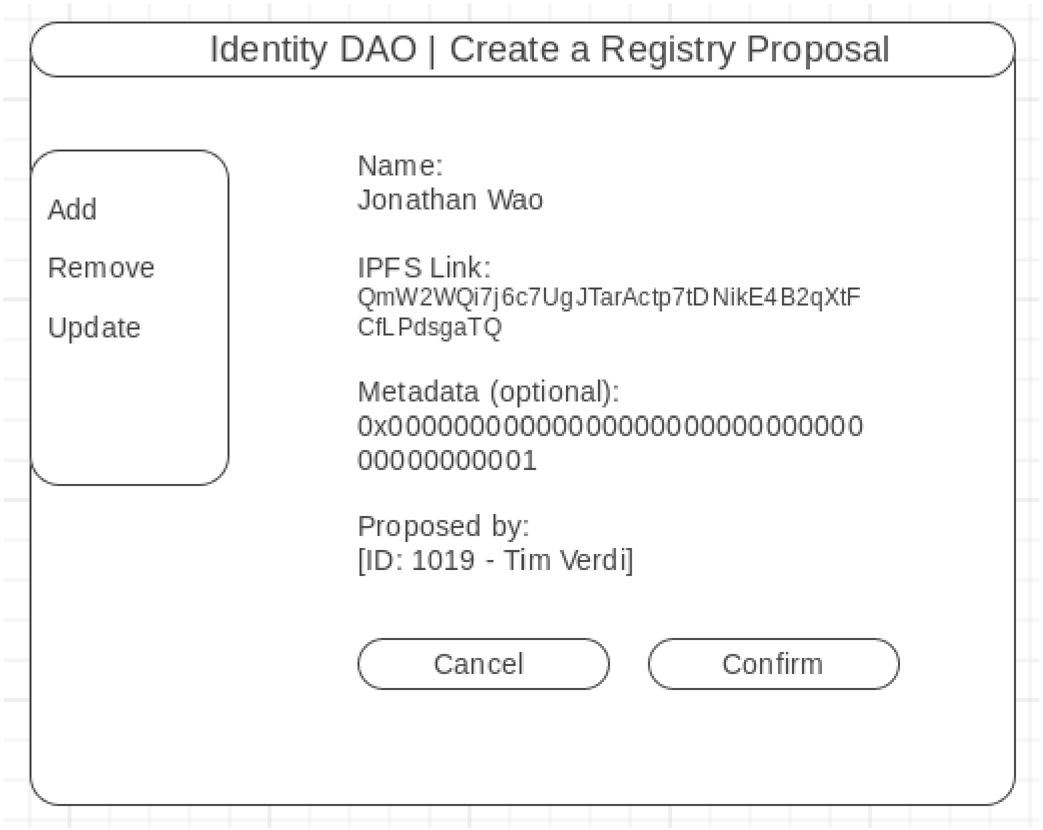
I. Identity DAO

For development convenience, an Identity DAO Javascript library may be provided to help verify users. This library should work as a straightforward direct interface to any actions which can be performed via the DAO itself.

Example of what using this library may look like:

```
const IdentityDAO = require('identity-dao');
const identityDAO = new IdentityDAO(web3);

IdentityDAO.isUnique('0xc1B1b64c33e0578DBa9E2CEacf0F8763128ddF63'); //returns a boolean
IdentityDAO.proposeAdd('Vitalik Buterin', '0x5E0318D57c2F0d1262df93478A92EeDAd246A374',
 'QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG'); //sends an addProposal, using the
 defaultAccount from web3. Alternate constructor inputs can be used to modify this
```

Example of a potential Identity DAO UI:



Name:
Jonathan Wao

IPFS Link:
QmW2WQi7j6c7UgJTarActp7tDNikE4B2qXtF
CfLPdsgaTQ

Metadata (optional):
0x00000000000000000000000000000
00000000001

Proposed by:
[ID: 1019 - Tim Verdi]

Note that the “Proposed by:” field is automatically filled based on the logged in user. This can work as additional visual confirmation on what the proposal will generate.

II. Identity DAO Plugin Integration

Making registered identities easy for dApps to verify and use is the optimal way to ensure Identity DAO's integration on various platforms. A simple script which developers can pop into the page and instantly get access to the ability to sign users may be a viable solution for this.

One potential userflow which works with Metamask may be:

- Identity DAO Javascript plugin attempts to pull web3 from **window.web3**
- Display a visible widget encouraging the user to click if signing in with Identity DAO
- On click of the widget…
    - The plugin checks if Metamask has been loaded. If not, notify the user and request unlocking Metamask. If not installed, link to basic instructions on installing Metamask.
    - If Metamask has been loaded, request a signature (prompted by Metamask). On receipt of the transaction hash, display a "loading" icon. If some failure occurs, display it.

The dApp is responsible for listening for this confirmed signature and handling session logic (e.g. timeout).

III. On-chain Architecture

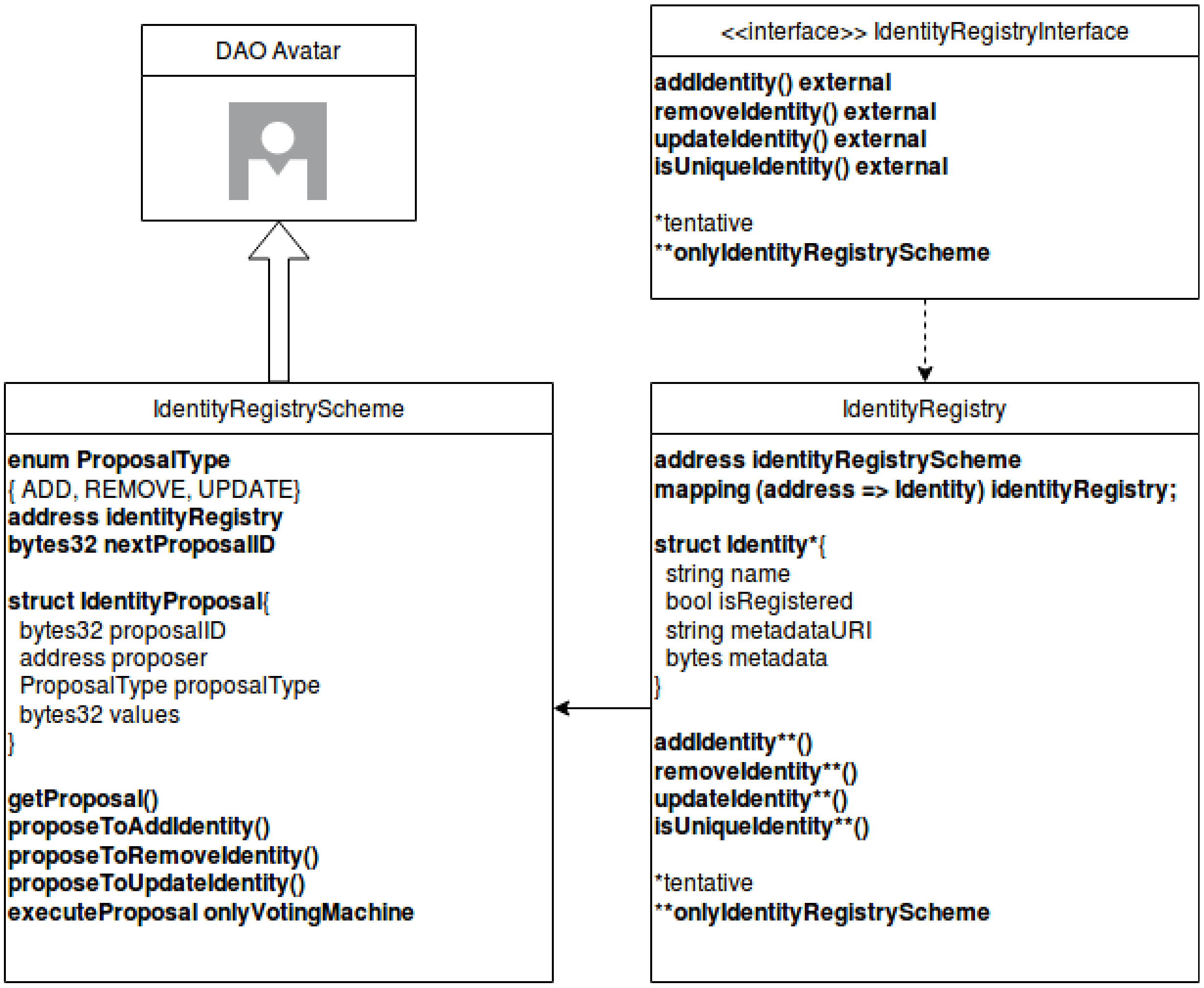
The smart contract architecture for the Identity DAO project is fairly simple; the DAO Avatar keeps control over the **IdentityRegistryScheme** contract, which acts as an interface for any validated user to propose adding, removing, or updating, a registered Identity. All proposals are stored within a mapping in the contract, as proposalIDs => IdentityProposal structs. Because the add, remove, and update functions all have different values of parameters, the struct simply holds these parameters as a single ABI-encoded value – bytes32.

On execution through the executeProposal() function, the contract will look up the proposal within the mapping, delete it (to prevent re-entrancy), and, based on the ProposalType enum value, destruct the values variable as expected.

The IdentityRegistryScheme contract, in turn, controls the **IdentityRegistry** itself. The IdentityRegistry contract can be seen as an isolated contract concerned only with the manipulation of the registry itself. In executing particular proposals, the Scheme contract will call the appropriate function on IdentityRegistry. These functions are restricted to the scheme contract, ensuring that DAO voting is required for using them.

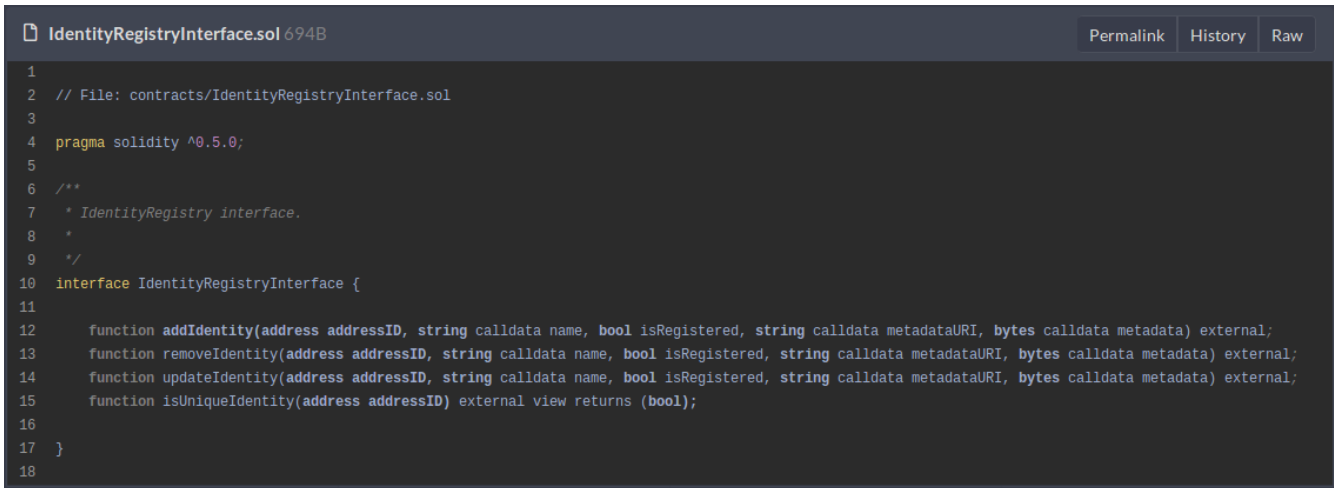
The Identity struct provides a boolean flag for determining registration, and a metadata URI linking to a JSON on IPFS with further detail on the identity. This can range from identity-related descriptions and contact information, to Twitter handles and special tags. The other two fields provide the opportunity to store a name and metadata directly on-chain, but is tentative, as this can also be stored directly on IPFS. Weighing which data may be preferable to keep on-chain vs. off-chain solutions will likely define the rest of these fields.

The next few pages will provide an overview of this architecture, alongside sample psuedo-code in Solidity.


DAO Avatar
<<interface>> IdentityRegistryInterface
addIdentity() external
removeIdentity() external
updateIdentity() external
isUniqueIdentity() external
*tentative
**onlyIdentityRegistryScheme
IdentityRegistryScheme
enum ProposalType
{ ADD, REMOVE, UPDATE}
address identityRegistry
bytes32 nextProposalID
struct IdentityProposal{
bytes32 proposalID
address proposer
ProposalType proposalType
bytes32 values
}
getProposal()
proposeToAddIdentity()
proposeToRemoveIdentity()
proposeToUpdateIdentity()
executeProposal onlyVotingMachine
IdentityRegistry
address identityRegistryScheme
mapping (address => Identity) identityRegistry;
struct Identity*{
string name
bool isRegistered
string metadataURI
bytes metadata
}
addIdentity**()
removeIdentity**()
updateIdentity**()
isUniqueIdentity**()
*tentative
**onlyIdentityRegistryScheme

Sample Code:

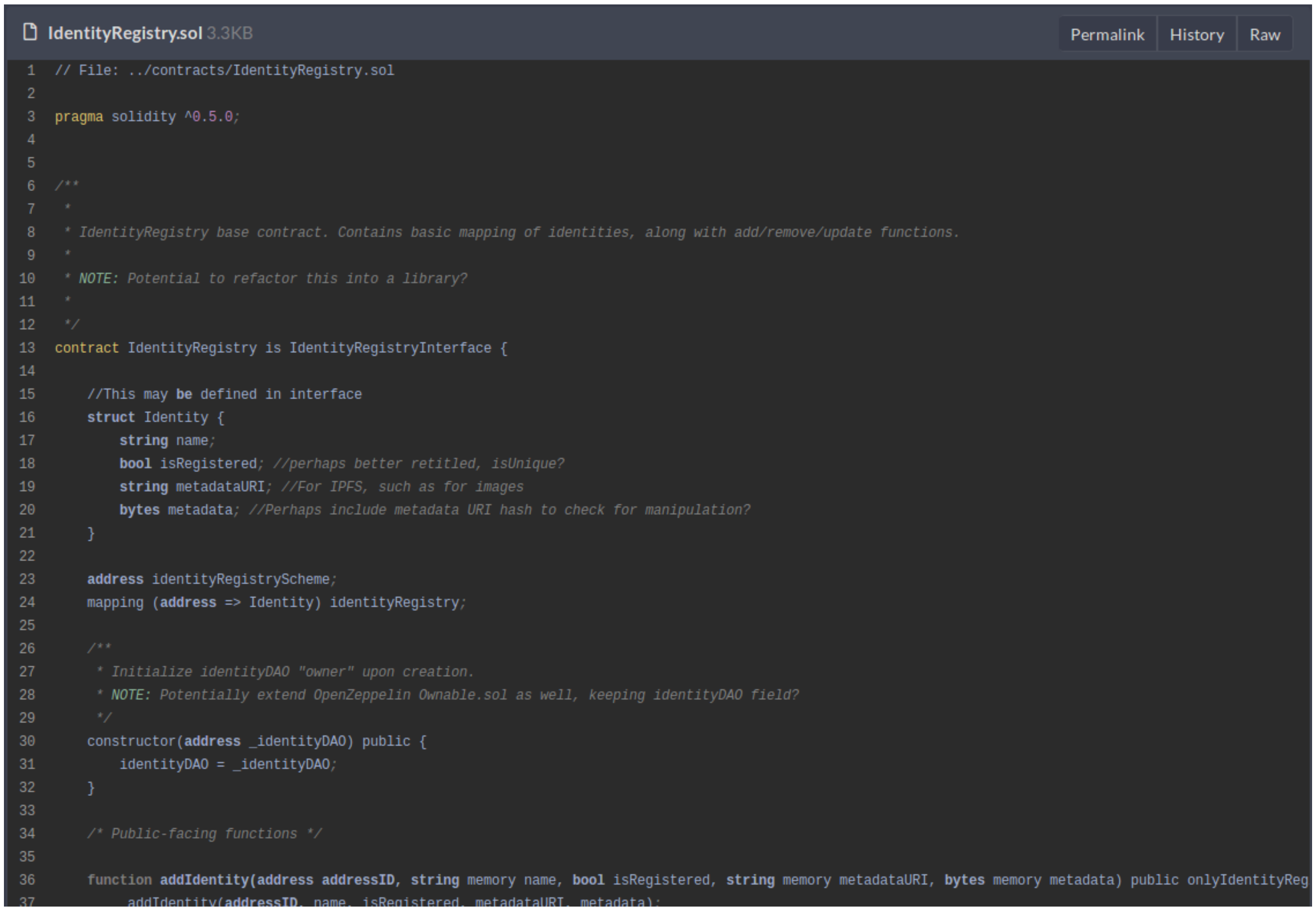
IdentityRegistryInterface.sol


IdentityRegistryInterface.sol 694B

Permalink History Raw

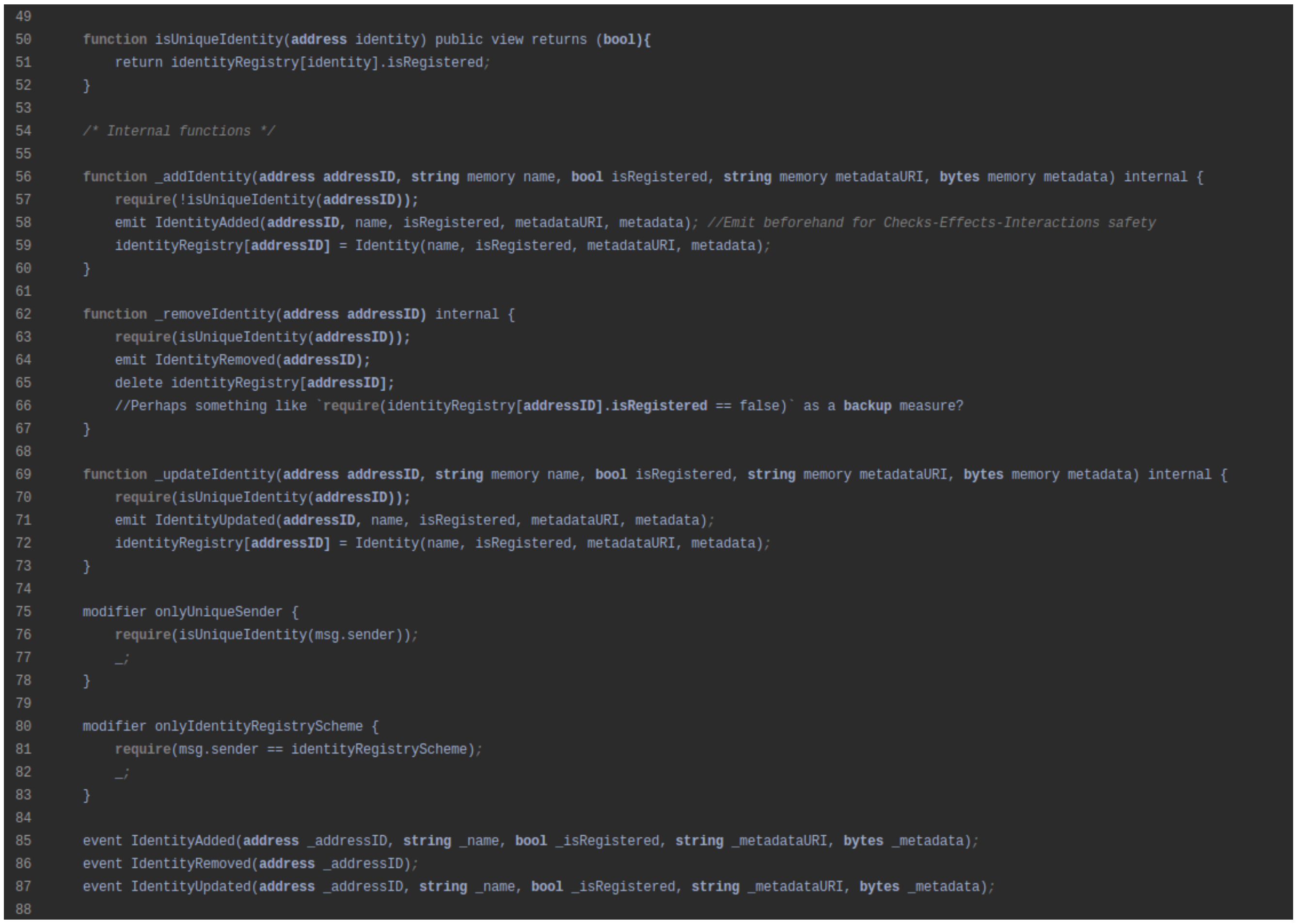
```

// File: contracts/IdentityRegistryInterface.sol

pragma solidity ^0.5.0;

/**
 * IdentityRegistry interface.
 *
 */
interface IdentityRegistryInterface {

    function addIdentity(address addressID, string calldata name, bool isRegistered, string calldata metadataURI, bytes calldata metadata) external;
    function removeIdentity(address addressID, string calldata name, bool isRegistered, string calldata metadataURI, bytes calldata metadata) external;
    function updateIdentity(address addressID, string calldata name, bool isRegistered, string calldata metadataURI, bytes calldata metadata) external;
    function isUniqueIdentity(address addressID) external view returns (bool);

}

```

IdentityRegistry.sol


IdentityRegistry.sol 3.3KB

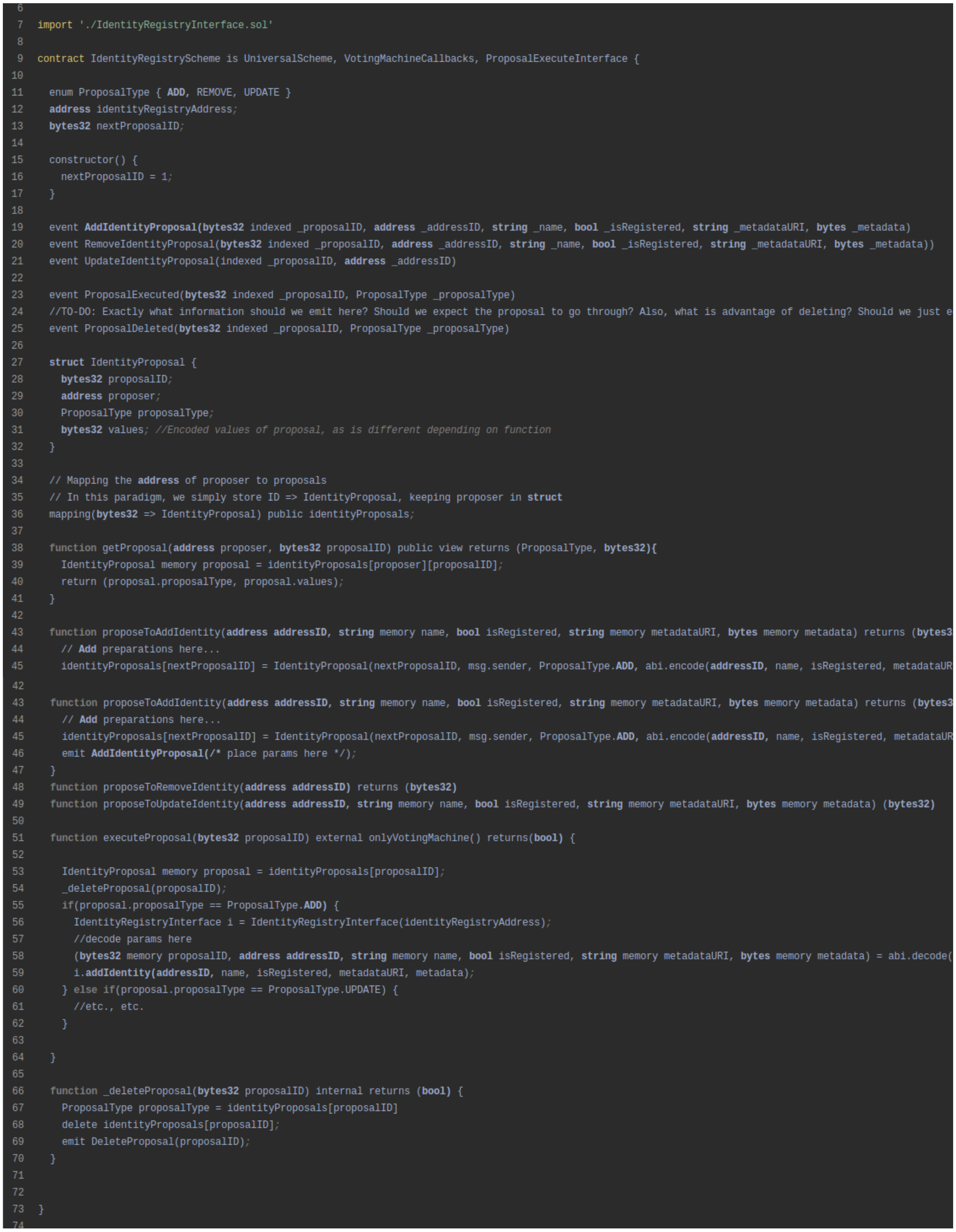
Permalink History Raw

```
// File: ../contracts/IdentityRegistry.sol

pragma solidity ^0.5.0;


/**
 *
 * IdentityRegistry base contract. Contains basic mapping of identities, along with add/remove/update functions.
 *
 * NOTE: Potential to refactor this into a library?
 *
 */
contract IdentityRegistry is IdentityRegistryInterface {

    //This may be defined in interface
    struct Identity {
        string name;
        bool isRegistered; //perhaps better retitled, isUnique?
        string metadataURI; //For IPFS, such as for images
        bytes metadata; //Perhaps include metadata URI hash to check for manipulation?
    }

    address identityRegistryScheme;
    mapping (address => Identity) identityRegistry;

    /**
     * Initialize identityDAO "owner" upon creation.
     * NOTE: Potentially extend OpenZeppelin Ownable.sol as well, keeping identityDAO field?
     */
    constructor(address _identityDAO) public {
        identityDAO = _identityDAO;
    }

    /* Public-facing functions */

    function addIdentity(address addressID, string memory name, bool isRegistered, string memory metadataURI, bytes memory metadata) public onlyIdentityReg
        addIdentity(addressID, name, isRegistered, metadataURI, metadata);
```



```

    function isUniqueIdentity(address identity) public view returns (bool){
        return identityRegistry[identity].isRegistered;
    }

    /* Internal functions */

    function _addIdentity(address addressID, string memory name, bool isRegistered, string memory metadataURI, bytes memory metadata) internal {
        require(!isUniqueIdentity(addressID));
        emit IdentityAdded(addressID, name, isRegistered, metadataURI, metadata); //Emit beforehand for Checks-Effects-Interactions safety
        identityRegistry[addressID] = Identity(name, isRegistered, metadataURI, metadata);
    }

    function _removeIdentity(address addressID) internal {
        require(isUniqueIdentity(addressID));
        emit IdentityRemoved(addressID);
        delete identityRegistry[addressID];
        //Perhaps something like `require(identityRegistry[addressID].isRegistered == false)` as a backup measure?
    }

    function _updateIdentity(address addressID, string memory name, bool isRegistered, string memory metadataURI, bytes memory metadata) internal {
        require(isUniqueIdentity(addressID));
        emit IdentityUpdated(addressID, name, isRegistered, metadataURI, metadata);
        identityRegistry[addressID] = Identity(name, isRegistered, metadataURI, metadata);
    }

    modifier onlyUniqueSender {
        require(isUniqueIdentity(msg.sender));
        _;
    }

    modifier onlyIdentityRegistryScheme {
        require(msg.sender == identityRegistryScheme);
        _;
    }

    event IdentityAdded(address _addressID, string _name, bool _isRegistered, string _metadataURI, bytes _metadata);
    event IdentityRemoved(address _addressID);
    event IdentityUpdated(address _addressID, string _name, bool _isRegistered, string _metadataURI, bytes _metadata);

```

IdentityRegistryScheme.sol



```
import './IdentityRegistryInterface.sol'

contract IdentityRegistryScheme is UniversalScheme, VotingMachineCallbacks, ProposalExecuteInterface {

  enum ProposalType { ADD, REMOVE, UPDATE }
  address identityRegistryAddress;
  bytes32 nextProposalID;

  constructor() {
    nextProposalID = 1;
  }

  event AddIdentityProposal(bytes32 indexed _proposalID, address _addressID, string _name, bool _isRegistered, string _metadataURI, bytes _metadata)
  event RemoveIdentityProposal(bytes32 indexed _proposalID, address _addressID, string _name, bool _isRegistered, string _metadataURI, bytes _metadata))
  event UpdateIdentityProposal(indexed _proposalID, address _addressID)

  event ProposalExecuted(bytes32 indexed _proposalID, ProposalType _proposalType)
  //TO-DO: Exactly what information should we emit here? Should we expect the proposal to go through? Also, what is advantage of deleting? Should we just e
  event ProposalDeleted(bytes32 indexed _proposalID, ProposalType _proposalType)

  struct IdentityProposal {
    bytes32 proposalID;
    address proposer;
    ProposalType proposalType;
    bytes32 values; //Encoded values of proposal, as is different depending on function
  }

  // Mapping the address of proposer to proposals
  // In this paradigm, we simply store ID => IdentityProposal, keeping proposer in struct
  mapping(bytes32 => IdentityProposal) public identityProposals;

  function getProposal(address proposer, bytes32 proposalID) public view returns (ProposalType, bytes32){
    IdentityProposal memory proposal = identityProposals[proposer][proposalID];
    return (proposal.proposalType, proposal.values);
  }

  function proposeToAddIdentity(address addressID, string memory name, bool isRegistered, string memory metadataURI, bytes memory metadata) returns (bytes3
    // Add preparations here...
    identityProposals[nextProposalID] = IdentityProposal(nextProposalID, msg.sender, ProposalType.ADD, abi.encode(addressID, name, isRegistered, metadataUR

  function proposeToAddIdentity(address addressID, string memory name, bool isRegistered, string memory metadataURI, bytes memory metadata) returns (bytes3
    // Add preparations here...
    identityProposals[nextProposalID] = IdentityProposal(nextProposalID, msg.sender, ProposalType.ADD, abi.encode(addressID, name, isRegistered, metadataUR
    emit AddIdentityProposal(/* place params here */);
  }
  function proposeToRemoveIdentity(address addressID) returns (bytes32)
  function proposeToUpdateIdentity(address addressID, string memory name, bool isRegistered, string memory metadataURI, bytes memory metadata) (bytes32)

  function executeProposal(bytes32 proposalID) external onlyVotingMachine() returns(bool) {

    IdentityProposal memory proposal = identityProposals[proposalID];
    _deleteProposal(proposalID);
    if(proposal.proposalType == ProposalType.ADD) {
      IdentityRegistryInterface i = IdentityRegistryInterface(identityRegistryAddress);
      //decode params here
      (bytes32 memory proposalID, address addressID, string memory name, bool isRegistered, string memory metadataURI, bytes memory metadata) = abi.decode(
      i.addIdentity(addressID, name, isRegistered, metadataURI, metadata);
    } else if(proposal.proposalType == ProposalType.UPDATE) {
      //etc., etc.
    }

  }

  function _deleteProposal(bytes32 proposalID) internal returns (bool) {
    ProposalType proposalType = identityProposals[proposalID]
    delete identityProposals[proposalID];
    emit DeleteProposal(proposalID);
  }


}
```