**// Provenance Contract**

// Contract that allow supply chain stakeholders to check the provenance of goods

pragma solidity ^0.4.24;

contract Provenance {

address CAuthority;

struct Producer {

string name;

string city;

uint contactno;

bool authorized;

}

struct Product {

address producer;

uint[] location; // array containing latitude & longitude

uint timeStamp;

}

mapping (address => Producer) producers;

mapping (string => Product) products;

function Provenance() {

CAuthority = msg.sender;

}

// Add producer (only CAuthority can add)

function Add\_Producer(string \_name, string \_city, uint \_contactno) onlyCAuthority returns (bool success) {

// Avoide overwrite of existing entries and ensure name is not null

if (bytes(producers[msg.sender].name).length == 0 && bytes(\_name).length != 0) {

producers[msg.sender].name = \_name;

producers[msg.sender].contactno = \_contactno;

producers[msg.sender].city = \_city;

producers[msg.sender].authorized = false;

return true;

}

else {

return false; // Entry already exists in database or a null name is entered

}

}

// Authorize producer (only CAuthority authorize)

function Authorize\_Producer(address \_producer) onlyCAuthority returns (bool success) {

producers [\_producer].authorized = true;

return true;

}

// Add product details (producer can add this)

function Add\_Product(string serial\_no, uint[] \_location) returns (bool success) {

// ensure no duplicate or null serial number

if (products[serial\_no].producer == 0X0 && bytes(serial\_no).length != 0) {

products [serial\_no].producer = msg.sender;

products[serial\_no].location = \_location;

products[serial\_no].timeStamp = block.timestamp;

return true;

}

else {

return false; // already used serial number or null entry

}

}

// Get producer details

function Find\_Producer(address \_producer) constant returns (string, uint, string, bool){

return (producers[\_producer].name, producers[\_producer].contactno, producers[\_producer].city, producers[\_producer].authorized);

}

// Get product details

function Find\_Product(string serial\_no) constant returns (address, uint[], uint) {

return (products[serial\_no].producer,

products [serial\_no] .location, products[serial\_no] .timeStamp);

}

// Remove producer (only CAuthority can remove)

function Remove\_Producer(address \_producer) onlyCAuthority returns (bool success) {

delete producers [\_producer];

return true;

}

// Remove product (only CAuthority can remove)

function Remove\_Product(string serial\_no) onlyCAuthority returns (bool success) {

delete products [serial\_no];

return true;

}

// Allow only CAuthority to execute specific functions using this modifier

modifier onlyCAuthority() {

if (msg.sender != CAuthority) throw;

\_;

}

}

**Modified 10 April 2019**

pragma solidity >=0.4.22 <0.6.0;

contract ProvenanceContract {

address CAuthority;

struct Producer {

string name;

string city;

uint contactno;

bool authorized;

}

struct Product {

address producer;

uint[] location; // array containing latitude & longitude

uint timeStamp;

}

mapping (address => Producer) producers;

mapping (string => Product) products;

function Provenance () public {

CAuthority = msg.sender;

}

// Add producer (only CAuthority can add)

function Add\_Producer(string memory \_name, string memory \_city, uint \_contactno) onlyCAuthority public returns (bool success) {

// Avoide overwrite of existing entries and ensure name is not null

if (bytes(producers[msg.sender].name).length == 0 && bytes(\_name).length != 0) {

producers[msg.sender].name = \_name;

producers[msg.sender].contactno = \_contactno;

producers[msg.sender].city = \_city;

producers[msg.sender].authorized = false;

return true;

}

else {

return false; // Entry already exists in database or a null name is entered

}

}

// Authorize producer (only CAuthority authorize)

function Authorize\_Producer(address \_producer) onlyCAuthority public returns (bool success) {

producers [\_producer].authorized = true;

return true;

}

// Add product details (producer can add this)

function Add\_Product(string memory serial\_no, uint[] memory \_location) public returns (bool success) {

// ensure no duplicate or null serial number

if (products[serial\_no].producer == 0x0000000000000000000000000000000000000000 && bytes(serial\_no).length != 0) {

products [serial\_no].producer = msg.sender;

products[serial\_no].location = \_location;

products[serial\_no].timeStamp = block.timestamp;

return true;

}

else {

return false; // already used serial number or null entry

}

}

// Get producer details

function Find\_Producer(address \_producer) public returns (string memory, uint, string memory, bool){

return (producers[\_producer].name, producers[\_producer].contactno, producers[\_producer].city, producers[\_producer].authorized);

}

// Get product details

function Find\_Product(string memory serial\_no) public returns (address, uint[] memory, uint) {

return (products[serial\_no].producer,

products [serial\_no] .location, products[serial\_no] .timeStamp);

}

// Remove producer (only CAuthority can remove)

function Remove\_Producer(address \_producer) onlyCAuthority public returns (bool success) {

delete producers [\_producer];

return true;

}

// Remove product (only CAuthority can remove)

function Remove\_Product(string memory serial\_no) onlyCAuthority public returns (bool success) {

delete products [serial\_no];

return true;

}

// Allow only CAuthority to execute specific functions using this modifier

modifier onlyCAuthority() {

if (msg.sender != CAuthority) revert();

\_;

}

}