TeamProject ERC-981

ERC-981 Partial Ownership Standard

1. บทนำ
   ERC-981 (Ethereum Request for Comments 981) เป็นมาตรฐานที่ถูกพัฒนาขึ้นสำหรับการกำหนดการเป็นเจ้าของบางส่วน (Partial Ownership) ของสินทรัพย์ดิจิทัลบน Ethereum เครือข่าย มาตรฐานนี้มีวัตถุประสงค์เพื่ออนุญาตให้หลายคนสามารถเป็นเจ้าของสินทรัพย์ดิจิทัลชิ้นเดียวกันได้ ซึ่งจะเป็นประโยชน์สำหรับการจัดการทรัพย์สินร่วมกัน เช่น ศิลปะดิจิทัล อสังหาริมทรัพย์เสมือน และสินค้าคงคลังต่างๆ

---

2. รายละเอียดเชิงเทคนิค (Technical Specification)
   ERC-981 กำหนดให้มีการปฏิบัติตามสัญญาอย่างน้อย 2 สัญญาคือ :
   - ERC721 หรือ ERC1155 สำหรับการระบุสินทรัพย์ (Asset)
   - ERC981PartialOwnership: เป็นสัญญาสำหรับการจัดการเจ้าของผู้ถือครองสินทรัพย์ (Asset owners)

   สัญญา ERC981PartialOwnership จะต้องมีฟังก์ชันต่อไปนี้:
   - ownerOf(uint256 assetId, uint256 timeTag) -> address[]:
     ส่งคืนรายการที่อยู่ของผู้ถือครองสินทรัพย์ในเวลาที่ระบุ
   - transfer(uint256 assetId, address[] callees, uint256[] timeTaggedValues): โอนการถือครองโดยระบุสัดส่วนการถือครองใหม่
   - transferBatch(uint256[] assetIds, address[] callees, uint256[] timeTaggedValues): โอนหลายสินทรัพย์ในคำสั่งเดียว
   - balanceOf(uint256 assetId, address owner, uint256 timeTag) -> uint256: ส่งคืนสัดส่วนการถือครองของเจ้าของที่กำหนดในเวลาที่ระบุ
   - balanceOfBatch(uint256[] assetIds, address[] owners, uint256[] timeTags) -> uint256[]: ส่งคืนสัดส่วนการถือครองของหลายสินทรัพย์

   นอกจากนี้ ยังมีเหตุการณ์ (Events) ที่จำเป็นต้องส่งออกเมื่อมีการโอนการถือครอง ได้แก่ Transfer และ TransferBatch

---

3. ตัวอย่างการนำไปใช้งานจริง
   การนำ ERC-981 ไปใช้งานจริงมีหลายกรณี เช่น:
   - ศิลปะดิจิทัล (Digital Art): ศิลปินสามารถสร้างผลงานศิลปะดิจิทัลแบบ Non-Fungible Token (NFT) และกำหนดสัดส่วนการเป็นเจ้าของให้กับผู้สนับสนุนหรือนักลงทุนต่างๆ ได้
   - อสังหาริมทรัพย์เสมือน (Virtual Real Estate): สามารถแบ่งการเป็นเจ้าของพื้นที่อสังหาริมทรัพย์เสมือนในโลกเสมือนจริง (Metaverse) ให้กับผู้เข้าร่วมหลายราย
   - สินค้าคงคลัง (Inventory): บริษัทสามารถแบ่งสัดส่วนการเป็นเจ้าของสินค้าคงคลังให้กับผู้มีส่วนได้ส่วนเสียต่างๆ เช่น ผู้ผลิต ผู้จัดจำหน่าย และลูกค้า เพื่อเพิ่มความโปร่งใสและประสิทธิภาพในการจัดการห่วงโซ่อุปทาน

4. โปรแกรม Solidity ตัวอย่างสำหรับ ERC-981 Token

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract ERC981PartialOwnership {
    // Mapping to store ownership data
    mapping(uint256 => mapping(uint256 => address[])) private _owners;
    mapping(uint256 => mapping(uint256 => mapping(address => uint256))) private _balances;

    // ERC721 token contract address
    ERC721 private _erc721Token;

    constructor(address erc721TokenAddress) {
        _erc721Token = ERC721(erc721TokenAddress);
    }

    /**
     * @dev Returns the list of owners for the given asset at the specified time.
     * @param assetId The ID of the asset.
     * @param timeTag The time to query the ownership data for.
     * @return A list of addresses representing the owners.
     */
    function ownerOf(uint256 assetId, uint256 timeTag) external view returns (address[] memory) {
        return _owners[assetId][timeTag];
    }

    /**
     * @dev Transfers ownership of the given asset by updating the ownership data.
     * @param assetId The ID of the asset to transfer ownership for.
     * @param callees The list of addresses to transfer ownership to.
     * @param timeTaggedValues The list of ownership fractions to transfer, each associated with a time tag.
     */
    function transfer(uint256 assetId, address[] calldata callees, uint256[] calldata timeTaggedValues) external {
        require(callees.length == timeTaggedValues.length, "Invalid input lengths");
        require(_erc721Token.ownerOf(assetId) == msg.sender, "Not the owner of the asset");

        uint256 totalTransferredValue = 0;
        for (uint256 i = 0; i < timeTaggedValues.length; i++) {
            _balances[assetId][timeTaggedValues[i]][callees[i]] += timeTaggedValues[i];
            _owners[assetId][timeTaggedValues[i]].push(callees[i]);
            totalTransferredValue += timeTaggedValues[i];
        }

        require(totalTransferredValue <= 1e18, "Total transferred value exceeds 100%");

        emit Transfer(assetId, msg.sender, callees, timeTaggedValues);
    }

    /**
     * @dev Transfers ownership of multiple assets in a batch operation.
     * @param assetIds The list of asset IDs to transfer ownership for.
     * @param callees The list of addresses to transfer ownership to.
     * @param timeTaggedValues The list of ownership fractions to transfer, each associated with a time tag.
     */
    function transferBatch(uint256[] calldata assetIds, address[] calldata callees, uint256[] calldata timeTaggedValues) external {
        require(assetIds.length == callees.length && callees.length == timeTaggedValues.length, "Invalid input lengths");

        for (uint256 i = 0; i < assetIds.length; i++) {
            transfer(assetIds[i], callees, timeTaggedValues);
        }

        emit TransferBatch(assetIds, msg.sender, callees, timeTaggedValues);
    }

    /**
     * @dev Returns the ownership fraction of the given owner for the specified asset at the given time.
     * @param assetId The ID of the asset.
     * @param owner The address of the owner.
     * @param timeTag The time to query the ownership data for.
     * @return The ownership fraction of the given owner.
     */
    function balanceOf(uint256 assetId, address owner, uint256 timeTag) external view returns (uint256) {
        return _balances[assetId][timeTag][owner];
    }

    /**
     * @dev Returns the ownership fractions of multiple owners for multiple assets in a batch operation.
     * @param assetIds The list of asset IDs.
     * @param owners The list of owner addresses.
     * @param timeTags The list of time tags to query the ownership data for.
     * @return The list of ownership fractions for the corresponding asset-owner-time combinations.
     */
    function balanceOfBatch(uint256[] calldata assetIds, address[] calldata owners, uint256[] calldata timeTags) external view returns (uint256[] memory) {
        require(assetIds.length == owners.length && owners.length == timeTags.length, "Invalid input lengths");

        uint256[] memory balances = new uint256[](assetIds.length);
        for (uint256 i = 0; i < assetIds.length; i++) {
            balances[i] = _balances[assetIds[i]][timeTags[i]][owners[i]];
        }

        return balances;
    }

    event Transfer(uint256 indexed assetId, address indexed from, address[] callees, uint256[] timeTaggedValues);
    event TransferBatch(uint256[] assetIds, address indexed from, address[] callees, uint256[] timeTaggedValues);
}
```

อธิบายในส่วนของโปรแกรม Partial Ownership Standard

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
```

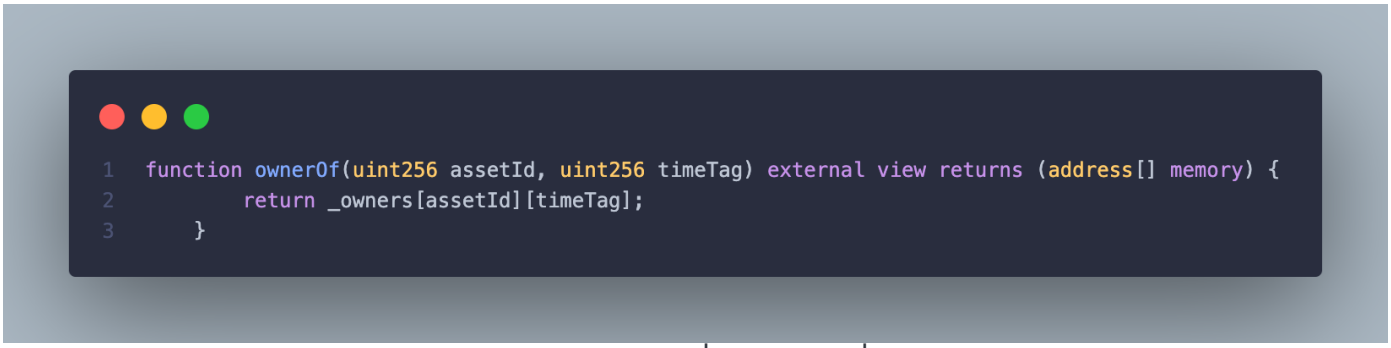
ส่วนนี้เป็นบรรทัดนำของสมาร์ทคอนแทรกต์ โดย SPDX-License-Identifier ระบุใบอนุญาตที่ใช้สำหรับโค้ดนี้ (MIT License) pragma ระบุเวอร์ชันของ Solidity compiler ที่ต้องการใช้ (เวอร์ชัน 0.8.0 หรือสูงกว่า) และ import นำเข้าสัญญา ERC721 จาก library OpenZeppelin ซึ่งให้การใช้งานมาตรฐาน ERC721 non-fungible token (NFT)

```
contract ERC981PartialOwnership {
    // Mapping to store ownership data
    mapping(uint256 => mapping(uint256 => address[])) private _owners;
    mapping(uint256 => mapping(uint256 => mapping(address => uint256))) private _balances;

    // ERC721 token contract address
    ERC721 private _erc721Token;

    constructor(address erc721TokenAddress) {
        _erc721Token = ERC721(erc721TokenAddress);
    }
```

ส่วนนี้กำหนดสัญญา ERC981PartialOwnership ซึ่งมีจุดประสงค์เพื่อใช้งานมาตรฐาน ERC-981 Partial Ownership สัญญานี้มี mapping 2 ตัว คือ _owners และ _balances _owners เป็น mapping ที่เชื่อมโยง assetId, timeTag และ array ของ addresses ของเจ้าของสินทรัพย์นั้นในเวลานั้น _balances เป็น mapping ที่เชื่อมโยง assetId, timeTag, address และสัดส่วนการถือครองของ address นั้นสำหรับสินทรัพย์นั้นในเวลานั้น สัญญายังมีตัวแปร _erc721Token ซึ่งเป็นตัวแทนของสัญญา ERC721 ที่ถูกกำหนดใน constructor ด้วยที่อยู่ของสัญญาโทเคน ERC721



```
function ownerOf(uint256 assetId, uint256 timeTag) external view returns (address[] memory) {
        return _owners[assetId][timeTag];
    }
```

ฟังก์ชัน ownerOf คือรายการของเจ้าของสำหรับสินทรัพย์ที่ระบุในเวลาที่กำหนด มันรับ assetId และ timeTag เป็นพารามิเตอร์ และคืนค่าเป็น array ของ addresses ของเจ้าของสินทรัพย์นั้นในเวลานั้น

```
function transfer(uint256 assetId, address[] calldata callees, uint256[] calldata timeTaggedValues) external {
        require(callees.length == timeTaggedValues.length, "Invalid input lengths");
        require(_erc721Token.ownerOf(assetId) == msg.sender, "Not the owner of the asset");

        uint256 totalTransferredValue = 0;
        for (uint256 i = 0; i < timeTaggedValues.length; i++) {
            _balances[assetId][timeTaggedValues[i]][callees[i]] += timeTaggedValues[i];
            _owners[assetId][timeTaggedValues[i]].push(callees[i]);
            totalTransferredValue += timeTaggedValues[i];
        }

        require(totalTransferredValue <= 1e18, "Total transferred value exceeds 100%");

        emit Transfer(assetId, msg.sender, callees, timeTaggedValues);
    }
```

ฟังก์ชัน transfer อนุญาตให้เจ้าของสินทรัพย์ถ่ายโอนสัดส่วนการถือครองไปยังหลาย addresses ในเวลาที่แตกต่างกันได้ มันรับ assetId, รายการ addresses (callees), และรายการสัดส่วนการถือครองพร้อมกับ timeTag (timeTaggedValues) ฟังก์ชันจะตรวจสอบความถูกต้องของความยาวอินพุตและว่าผู้เรียกใช้เป็นเจ้าของสินทรัพย์หรือไม่ จากนั้นจะวนลูปผ่าน timeTaggedValues อัปเดต _balances ด้วยสัดส่วนการถือครองใหม่และเพิ่ม owners ใหม่ใน _owners ฟังก์ชันจะตรวจสอบว่ามูลค่ารวมที่ถ่ายโอนไม่เกิน 100% และจะปล่อยอีเวนท์ Transfer ออกมาพร้อมด้วยข้อมูลที่เกี่ยวข้อง

```
function transferBatch(uint256[] calldata assetIds, address[] calldata callees, uint256[] calldata timeTaggedValues) external {
        require(assetIds.length == callees.length && callees.length == timeTaggedValues.length, "Invalid input lengths");

        for (uint256 i = 0; i < assetIds.length; i++) {
            transfer(assetIds[i], callees, timeTaggedValues);
        }

        emit TransferBatch(assetIds, msg.sender, callees, timeTaggedValues);
    }
```

ฟังก์ชัน transferBatch คล้ายกับ transfer แต่มีความสามารถในการถ่ายโอนสัดส่วนการถือครองของหลายสินทรัพย์ในการทำธุรกรรมเดียว มันรับ รายการของ assetIds (assetIds), รายการ addresses (callees), และรายการสัดส่วนการถือครองพร้อมกับ timeTag (timeTaggedValues) ฟังก์ชันจะตรวจสอบความยาวของอินพุตก่อน จากนั้นจะเรียกใช้ transfer สำหรับแต่ละ assetId โดยส่งรายการ callees และ timeTaggedValues เดียวกัน และในตอนท้ายจะปล่อยอีเวนท์ TransferBatch ออกมาพร้อมข้อมูลที่เกี่ยวข้อง

```
function balanceOf(uint256 assetId, address owner, uint256 timeTag) external view returns (uint256) {
        return _balances[assetId][timeTag][owner];
    }
```

ฟังก์ชัน balanceOf คืนค่าสัดส่วนการถือครองของเจ้าของที่ระบุสำหรับสินทรัพย์ที่ระบุ ในเวลาที่กำหนด มันรับ assetId, owner และ timeTag เป็นพารามิเตอร์ และคืนค่าสัดส่วนการถือครองที่ตรงกับการผสมของ assetId, owner และ timeTag จากแมพ _balances

```
function balanceOfBatch(uint256[] calldata assetIds, address[] calldata owners, uint256[] calldata timeTags) external view returns (uint256[] memory) {
        require(assetIds.length == owners.length && owners.length == timeTags.length, "Invalid input lengths");

        uint256[] memory balances = new uint256[](assetIds.length);
        for (uint256 i = 0; i < assetIds.length; i++) {
            balances[i] = _balances[assetIds[i]][timeTags[i]][owners[i]];
        }

        return balances;
    }
```

ฟังก์ชัน balanceOfBatch เป็นเวอร์ชันการทำงานแบบกลุ่มของ balanceOf มันรับรายการ assetIds (assetIds), รายการ owners (owners) และรายการ timeTags (timeTags) ฟังก์ชันจะตรวจสอบความยาวของอินพุตก่อน จากนั้นจะสร้างอาร์เรย์ balances เพื่อเก็บสัดส่วนการถือครอง จากนั้นจะวนลูปผ่านรายการอินพุต เพื่อดึงสัดส่วนการถือครองจากแมพ _balances และเก็บไว้ในอาร์เรย์ balances ตามลำดับ และคืนค่าอาร์เรย์ balances ในตอนท้าย

```
event Transfer(uint256 indexed assetId, address indexed from, address[] callees, uint256[] timeTaggedValues);
event TransferBatch(uint256[] assetIds, address indexed from, address[] callees, uint256[] timeTaggedValues);
```

ส่วนสุดท้ายนี้เป็นการประกาศอีเวนท์ Transfer และ TransferBatch ที่จะถูกปล่อยออกมาเมื่อมีการโอนสัดส่วนการถือครองของสินทรัพย์ อีเวนท์เหล่านี้ประกอบด้วยข้อมูลที่เกี่ยวข้องกับการโอน เช่น assetId, ผู้โอน, รายการผู้รับโอน และรายการสัดส่วนการถือครองที่โอนพร้อมกับ timeTag