ITAM GAMES Digital Assets V 0.1

Introducation

1개의 게임사 혹은 회사에서 N개의 게임 혹은 앱의 디지털 자산을 만들 수 있도록 설계 되어있습니다. 그렇기 때문에 symbol은 게임 혹은 앱의 고유한 키값이 되며, 그에 따른 1개의 symbol에 여러개의 디지털 자산이 만들어 질 수 있습니다. 또한 RAM의 소모가 있긴 하지만 모든 디지털 자산의 상세 정보가 테이블, 블록에 등록되며, 해당 디지털 자산의 히스토리를 위해 등록, 수정, 삭제가 블록에 등록 됩니다.(각각의 action마다 reason또는 memo가 있습니다.)

Required Methods

CREATE

게임별 디지털 자산을 등록 합니다. 그에 따라 심볼은 게임별로 유니크한 값이 됩니다. 해당 게임의 카테고리를 등록하여, 어떠한 아이템이 게임 안에서 사용되는지를 확인 할 수 있고 나아가 디지털 자산별 유효성 체크의 기준이 될 수 있습니다. symbol은 A-Z로 최대 7자리 가능합니다.

```
/*
@param name     issuer: 디지털 자산 발행자
@param string   symbol_name: 디지털 자산 심볼 이름
@param uint64_t app_id: 앱 아이디
@param string   structs: 발행될 디지털 자산의 구조 (Stringified Json)
structs Example
[
    {
        "category": "weapon",
        "fields": ["str", "dex", "luk"]
    }
]
*/
ACTION create(name issuer, string symbol_name, uint64_t app_id, string
structs)
```

> 카테고리가 필요한 이유는 잘못 발행되는 디지털 자산을 막기 위한 최소한의 조치 입니다.

ISSUE

이슈 메소드는 디지털 자산을 발행하고 소유권을 'to' 계정 이름으로 지정합니다. 해당 메소드는 create 메소드를 먼저 호출 후에 사용 될 수 있습니다. fungible이 아닌 디지털 자산일 경우 quantity을 1개로 해야되며, fungible일 경우에는 1보다 크거나 같아야됩니다. 디지털 자산의 히스토리를 위해 발행 사유가 포함되어야 됩니다.

```
/*
```

```
@param name     to: 디지털 자산 소유자
@param quantity quantity: 발행할 디지털 자산 개수
@param uint64_t token_id: 디지털 자산 아이디
@param string   token_name: 디지털 자산 이름
@param string   category: 발행할 디지털 자산의 카테고리
@param string   options: 발행할 디지털 자산의 옵션 (Stringified Json)
@param bool     fungible: fungible 디지털 자산 여부
@param string   reason: 디지털 자산의 발행 사유

options Example
{
    "str": 1,
    "dex": 2,
    "luk": 3
}
*/
ACTION issue(name to, asset quantity, uint64_t token_id, string token_name,
string category, bool fungible, string options, string reason)
```

> create에 등록된 카테고리가 아닌경우 발행시 에러가 납니다.

TRANSFERNFT

non-fungible 디지털 자산을 전송하는 데 사용됩니다. 여러개의 디지털 자산을 전송 가능하며, 디지털 자산을 소유자만이 실행 할 수 있습니다.

```
/*
@param name     from: 디지털 자산을 보내는 유저
@param name     to: 디지털 자산을 받는 유저
@param string   symbol_name: 심볼 이름
@param vector<uint64_t> token_ids: 디지털 자산 아이디
@param string   memo: 메모
*/
ACTION transfernft(name from, name to, string symbol_name, vector<uint64_t>
token_ids, string memo);
```

> 메모에는 해당 디지털 자산의 이동에 대한 히스토리를 남길 수 있습니다.

TRANSFER

eosio.token에서 사용되는 표준 transfer메소드는 fungible 디지털 자산을 transfer 합니다. 기존 eosio.token과 다른 점으로는 디지털 자산을 구분 할 수 있는 token_id가 추가 되었습니다.

```
/*
@param name     from: 디지털 자산을 보내는 유저
@param name     to: 디지털 자산을 받는 유저
@param asset    quantity: 디지털 자산 asset
@param uint64_t token_id: 디지털 자산 아이디
@param string   memo: 메모
*/
ACTION transfer(name from, name to, asset quantity, uint64_t token_id,
string memo)
```

> 메모에는 해당 디지털 자산의 이동에 대한 히스토리를 남길 수 있습니다.

BURN

fungible 디지털 자산을 삭제 하는데 사용됩니다. 디지털 자산의 모든 수량을 삭제하면 사용된 RAM이 회수됩니다. 디지털 자산의 히스토리를 위해서 삭제 사유를 입력해야됩니다. 디지털 자산을 소유자와 해당 게임사만 실행 할 수 있습니다.

```
/*
@param name     owner: 디지털 자산 소유자
@param asset    quantity: 디지털 자산 asset
@param uint64_t token_id: 디지털 자산 아이디
@param string   reason: 삭제 사유
*/
ACTION burn(name owner, asset quantity, uint64_t token_id, string reason)
```

> 소유자와 게임사가 실행 가능한 이유는 게임을 하는 도중에 소유자가 필요 없다고 생각이 되어 파괴를 할 수 있으며, 또한 게임사에서도 소유자의 아이템에 대해서 무언가의 엑션이 있을때 삭제 할 수 있어야 됩니다.(예. 아이템 강화 실패 등등)

BURNNFT

non-fungible 디지털 자산을 삭제 하는데 사용됩니다. 여러개의 디지털 자산을 삭제 가능하며, 디지털 자산을 삭제하면 사용된 RAM이 회수됩니다. 디지털 자산의 히스토리를 위해서 삭제 사유를 입력해야됩니다. 디지털 자산을 소유자와 게임사만 실행 할 수 있습니다.

```
/*
@param name     owner: 디지털 자산 소유자
@param string   symbol_name: 디지털 자산 symbol
@param vector<uint64_t> token_ids: 디지털 자산 아이디들
@param string   reason: 삭제 사유
*/
ACTION burnnft(name owner, string symbol_name, vector<uint64_t> token_ids,
string reason)
```

> 소유자와 게임사가 실행 가능한 이유는 게임을 하는 도중에 소유자가 필요 없다고 생각이 되어 파괴를 할 수 있으며, 또한 게임사에서도 소유자의 아이템에 대해서 무언가의 엑션이 있을때 삭제 할 수 있어야 됩니다.(예. 아이템 강화 실패 등등)

ADDCATEGORY

발행 가능한 디지털 자산의 카테고리를 추가합니다.

```
/*
@param string   symbol_name: 디지털 자산의 심볼
@param string   category_name: 카테고리 이름
@param string[] fields: 카테고리의 필수 필드
*/
ACTION addcategory(string symbol_name, string category_name, vector<string>
fields)
```

> 게임 특성상 시간이 지날 수록 신규 아이템이 만들어져야됩니다. 그에 따라 카테고리를 추가 할 수 있어야됩니다.

MODIFY

소유자의 디지털 자산의 이름과 옵션을 수정합니다. 디지털 자산의 히스토리를 위해서 수정 사유를 입력해야됩니다. 해당 메소드는 게임사만 실행 가능합니다.

```
/*
@param name     owner: 디지털 자산 소유자
@param string   symbol_name: 심볼 이름
@param uint64_t token_id: 디지털 자산 아이디
@param string   token_name: 수정할 디지털 자산 이름
@param string   options: 수정할 옵션 (Stringified Json)
@param string   reason: 수정 사유

option Example
{
    "str": 100,
    "dex": 200,
    "luk": 300
}
*/

ACTION modify(name owner, string symbol_name, uint64_t token_id, string
token_name, string options, string reason)
```

> 게임을 하는 도중 디지털 자산에 대한 엑션이 있을 경우(예. 강화 성공, 강화 실패 등등) 디지털 자산의 수정이 필요합니다. 그래서 게임사의 권한으로 디지털 자산의 수정을 할 수 있습니다.

Token Data

Currency Table

게임별 디지털 자산에 대한 정보를 저장하는 테이블입니다. supply는 토큰이 발행될 때마다 자동으로 증가합니다. categories에 정의된 유형으로만 디지털 자산 발행이 가능합니다.

```
TABLE currency
{
    name issuer;
    asset supply;
    uint64_t app_id;
    vector<category> categories;
};


struct category
{
    string name;
    vector<string> fields;
};
```

Account Table

디지털 자산의 상세 정보를 저장하고 있는 테이블입니다. balance는 현재 가지고 있는 디지털 자산의 개수입니다.

```
TABLE account
{
    asset balance;
    map<uint64_t, token> tokens;
};


struct token
{
    string category;
    string token_name;
    bool fungible;
    uint64_t count;
    string options; // JSON.stringify
};
```