ITAM GAMES Digital Assets V 0.1.1

Introducation

This contract is structured so that a single gaming company could make digital assets on 'N' number of games or apps. Therefore, the symbol would be the unique key for a game or an app, and thus multiple digital assets can be created under one symbol. Also, although there are costs for RAM, all of the details of the digital assets are registered in both block and table. Registration, modification, and deletion of assets are recorded on block for the history of digital assets. (For every action, there is a 'reason' and/or 'memo')

Required Methods

CREATE

Digital assets are registered by games. In accordance to this, the symbols are unique for each game. By registering the category of the game, users can check which items are used in the game. Furthermore, it can also be used as a basis for authenticating the validity of each digital assets. Symbol can be up to 7 digits in A-Z.

```
/*
@param name             issuer: Digital assets issuer
@param symbol_code      symbol_name: Digital assets symbol name
@param uint64_t         app_id: App ID
@param string           structs: Structure of digital assets (Stringified Json)
structs Example
[
    {
        "category": "weapon",
        "fields": ["str", "dex", "luk"]
    }
]
*/
ACTION create(name issuer, symbol_code symbol_name, uint64_t app_id, string
structs)
```

> The category is needed to serve as the minimal measure to prevent mis-issued digital assets.

ISSUE

For the Issue method, assign ownership to the 'to' account name after issuing the digital assets. This method can only be used after the create method is called. For non-fungible digital assets, the quantity must be one, and for fungible assets, the quantity must be greater than or equal to 1. The reason for issuance must be included for the history of digital assets.

```
/*
@param name     to: Digital assets owner
@param asset    quantity: Number of digital assets to issue
@param string   token_name: Digital assets name
@param string   category: Category for digital assets to be issued
@param string   options: Option for digital assets to be issued (Stringified
Json)
@param bool     fungible: Digital assets type, fungible vs non-fungible
@param string   reason: Reason for issuance

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

> If the category is unregistered on create, error will occur

TRANSFERNFT

Used to transfer non-fungible digital assets. Can transfer multiple digital assets, and can only be executed by the digital asset owner.

```
/*
@param name             from: User who sends digital assets
@param name             to: User who receives digital assets
@param symbol_code      symbol_name: Symbol name
@param vector<uint64_t> token_ids: Digital asset IDs
@param string   memo: memo
*/
ACTION transfernft(name from, name to, symbol_code symbol_name,
vector<uint64_t> token_ids, string memo);
```

> Transaction history of digital assets can be recorded on memo.

TRANSFER

The standard transfer method used by eosio.token are for transferring fungible digital assets. It's different from the existing eosio.token in that token_id has been added to distinguish digital assets.

```
/*
@param name     from: User who sends digital assets
@param name     to: User who receives digital assets
@param asset    quantity: Digital assets type
@param uint64_t token_id: Digital assets ID
@param string   memo: mome
*/
ACTION transfer(name from, name to, asset quantity, uint64_t token_id, string
memo)
```

> Transaction history of digital assets can be recorded on memo.

BURN

Used to delete fungible assets. If all quantities of the digital assets are deleted, the RAM used is reclaimed. The reason for deletion is needed for the history of digital assets. Only the digital assets owner and the corresponding game company can execute this.

```
/*
@param name     owner: Digital assets owner
@param asset    quantity: Digital assets quantity
@param uint64_t token_id: Digital assets id
@param string   reason: Reason for deletion
*/
ACTION burn(name owner, asset quantity, uint64_t token_id, string reason)
```

> The owner and the game company are able to execute because (1) the owner may feel that he/she no longer needs the item and thus burn, and (2) the game company must be able to delete when there is an action in accordance to the owner's item (EX. Failure of item enhancement, etc.)

BURNNFT

Used to delete non-fungible digital assets. Multiple digital assets can be deleted, and the RAM is reclaimed once the digital assets are deleted. The Reason for deletion is needed for the history of digital assets. Only the digital asset owner and game company can execute this.

```
/*
@param name              owner: Digital assets owner
@param symbol_code       symbol_name: Digital assets symbol
@param vector<uint64_t> token_ids: Digital assets IDs
@param string            reason: Reason for deletion
*/
ACTION burnnft(name owner, symbol_code symbol_name, vector<uint64_t>
token_ids, string reason)
```

> The owner and the game company are able to execute because (1) the owner may feel that he/she no longer needs the item and thus burn, and (2) the game company must be able to delete when there is an action in accordance to the owner's item (EX. Failure of item enhancement, etc.)

ADDCATEGORY

add categories of publishable digital assets.

```
/*
@param symbol_code     symbol_name: Digital assets symbol
@param string          category_name: Category name
@param string[]        fields: field of category
*/
ACTION addcategory(symbol_code symbol_name, string category_name,
vector<string> fields)
```

> Due to the nature of the game, new items need to be created over time. Thus, categories should be to addable accordingly.

MODIFY

Modify the name and options of the owner's digital assets. The reason for modification is needed for the history of the digital assets. This method can only be executed by the gaming company.

```
/*
@param name            owner: Digital assets owner
@param symbol_code     symbol_name: Symbol name
@param uint64_t        token_id: Digital assets id
@param string          token_name: Name of digital assets to be modified
@param string          options: Options to be modified (Stringified Json)
@param string          reason: reason for modification

option Example
{
    "str": 100,
    "dex": 200,
```

```
    "luk": 300
}
*/

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

> If there is an action on the digital assets while playing the game (e.g. successful enhancement, enhancement failure, etc.), digital asset modification is needed. That is why gaming companies are given the authorization to modify the digital assets information.

Token Data

Currency Table

This is the table that stores the info of the digital assets by games. Supply will automatically increase each time a token is issued. Digital assets can only be issued by the types defined in categories.

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


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

Account Table

This is table that stores the detailed information of digital assets. Balance is the number of digital assets currently owned.

```
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
};
```