Non-fungible token LLD

Introduction

This document outlines high level design of non-fungible token support in Peerplays blockchain.

Non-fungible tokens (NFTs) are unique, digital items with blockchain-managed ownership. Examples include collectibles, game items, digital art, event tickets, domain names, and even ownership records for physical assets.

Our design will relly on EIP-721 standard, originating from Ethereum blockchain.

Original EIP can be found here:
https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md

We will transfer concepts from EIP to the Peerplays to the maximum possible extent. Due to the different natures of Ethereum and Peerplays, it will not be possible to make 1 on 1 matches for all features.

Many parts of this document will use infoprmation from original document, available here:
https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md

Abstract

We considered use cases of NFTs being owned and transacted by individuals as well as consignment to third party brokers/wallets/auctioneers ("operators"). NFTs can represent ownership over digital or physical assets. We considered a diverse universe of assets, and we know you will dream up many more:

Physical property — houses, unique artwork

Virtual collectables — unique pictures of kittens, collectable cards

"Negative value" assets — loans, burdens and other responsibilities

In general, all houses are distinct and no two kittens are alike. NFTs are distinguishable and you must track the ownership of each one separately.

NFT representation

```
class nft_object : public abstract_object<nft_object>
{
   public:
      static const uint8_t space_id = protocol_ids;
      static const uint8_t type_id = nft_object_type;

      account_id_type  owner;
      vector<account_id_type> approved_operators;

      string           metadata;
};

struct by_owner;
struct by_owner_and_id;
using nft_multi_index_type = multi_index_container<
   nft_object,
   indexed_by<
      ordered_unique< tag<by_id>,
         member<object, object_id_type, &object::id>
      >,
      ordered_non_unique< tag<by_owner>,
         member<nft_object, account_id_type, &nft_object::owner>
      >,
      ordered_unique< tag<by_owner_and_id>,
         composite_key<nft_object,
            member<nft_object, account_id_type, &nft_object::owner>,
            member<object, object_id_type, &object::id>
         >
      >
   >
>;
using nft_index = generic_index<nft_object, nft_multi_index_type>;
```

NFT operations

balance_of(account_id_type owner)

```
// matches balanceOf(address _owner)

// This method does not change chain state and does not require
evaluator

// get nft_index by_owner
// use count method with owner as param
```

owner_of(nft_id_type token_id)

```
// matches function ownerOf(uint256 _tokenId)

// This method does not change chain state and does not require
evaluator

// get nft_index by_id
// to retrieve nft_object, use find method with token_id as param
// return nft_object.owner
```

safe_transfer_from(account_id_type from, account_id_type to, nft_id_type token_id, string data)

```
// matches function safeTransferFrom(address _from, address _to,
uint256 _tokenId, bytes data)

// for do_evaluate
// get nft_index by_id
// to retrieve nft_object, use find method with token_id as param
// check if nft_object with id token_id exists in nft_index by_id
// get account_index by_id
// to retrieve owner account_object, use find method with nft_object.
owner as param
// to retrieve from account_object, use find method with from as param
// to retrieve to account_object, use find method with to as param
// check if payer is same account as nft_object.owner or is in the
approved_operators list
// check if from is same account as nft_object.owner
// check if account object with id to exists in account_index by_id

// for do_apply
// get nft_index by_id
// to retrieve nft_object, use find method with token_id as param
// set nft_object.owner field to the value of to parameter
// clear the approved_operator list, as the owner is changed

// Must have non zero fee
```

safe_transfer_from(account_id_type from, account_id_type to, nft_id_type token_id)

```
// matches function safeTransferFrom(address _from, address _to,
uint256 _tokenId)

// We will not implement this method separately
// We will just declare that the default value for data parameter is
empty string
```

transfer_from(account_id_type from, account_id_type to, nft_id_type token_id)

```
// matches function transferFrom(address _from, address _to, uint256
_tokenId)

// This function will be implemented as alias of safe_transfer_from
(account_id_type from, account_id_type to, nft_id_type token_id)
// The check mentioned in function description does not apply to
Peerplays
```

approve(account_id_type approved, nft_id_type token_id)

```
// matches function approve(address _approved, uint256 _tokenId)

// for do_evaluate
// get nft_index by_id
// to retrieve nft_object, use find method with token_id as param
// check if nft_object with id token_id exists in nft_index by_id
// get account_index by_id
// to retrieve owner account_object, use find method with nft_object.
owner as param
// to retrieve approved account_object, use find method with approved
as param
// check if payer is same account as nft_object.owner or is in the
approved_operators list
// check if account object with id approved exists in account_index
by_id

// for do_apply
// get nft_index by_id
// to retrieve nft_object, use find method with token_id as param
// add approved to nft_object.approved_operators list

// Must have non zero fee
```

set_approval_for_all(account_id_type operator, bool approved)

```
// matches function setApprovalForAll(address _operator, bool _approved)

// for do_evaluate
// get account_index by_id
// check if account object with id operator exists in account_index
by_id

// for do_apply
// get nft_index by_owner
// iterate through the index filtered by op.payer, and for each token
// - if approved is true, add operator to the list of approved operators
// - if approved is false, remove operator from the list of approved
operators, if exists

// Must have non zero fee
```

get_approved(nft_id_type token_id)

```
// matches function getApproved(uint256 _tokenId)

// This method does not change chain state and does not require
evaluator

// get nft_index by_id
// to retrieve nft_object, use find method with token_id as param
// return first element in approved_operators, or null address
```

is_approved_for_all(account_id_type owner, account_id_type operator)

```
// matches function isApprovedForAll(address _owner, address _operator)

// This method does not change chain state and does not require
evaluator

// get nft_index by_owner
// if index filtered by owner has no elements, return false
// set result variable to true
// iterate through the index filtered by owner, and for each token
// - result = result && (operator is in the list of approved_operators)
// return result variable
```