Non-fungible token HLD

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.

Specification

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Every ERC-721 compliant contract must implement the ERC721 and ERC165 interfaces (subject to "caveats" below):

```
pragma solidity ^0.4.20;

/// @title ERC-721 Non-Fungible Token Standard
/// @dev See https://eips.ethereum.org/EIPS/eip-721
///  Note: the ERC-165 identifier for this interface is 0x80ac58cd.
interface ERC721 /* is ERC165 */ {
    /// @dev This emits when ownership of any NFT changes by any
mechanism.
    ///  This event emits when NFTs are created (`from` == 0) and
destroyed
    ///  (`to` == 0). Exception: during contract creation, any number
of NFTs
    ///  may be created and assigned without emitting Transfer. At the
time of
    ///  any transfer, the approved address for that NFT (if any) is
reset to none.
    event Transfer(address indexed _from, address indexed _to, uint256
indexed _tokenId);
```

```
    /// @dev This emits when the approved address for an NFT is changed
or
    ///  reaffirmed. The zero address indicates there is no approved
address.
    ///  When a Transfer event emits, this also indicates that the
approved
    ///  address for that NFT (if any) is reset to none.
    event Approval(address indexed _owner, address indexed _approved,
uint256 indexed _tokenId);

    /// @dev This emits when an operator is enabled or disabled for an
owner.
    ///  The operator can manage all NFTs of the owner.
    event ApprovalForAll(address indexed _owner, address indexed
_operator, bool _approved);

    /// @notice Count all NFTs assigned to an owner
    /// @dev NFTs assigned to the zero address are considered invalid,
and this
    ///  function throws for queries about the zero address.
    /// @param _owner An address for whom to query the balance
    /// @return The number of NFTs owned by `_owner`, possibly zero
    function balanceOf(address _owner) external view returns (uint256);

    /// @notice Find the owner of an NFT
    /// @dev NFTs assigned to zero address are considered invalid, and
queries
    ///  about them do throw.
    /// @param _tokenId The identifier for an NFT
    /// @return The address of the owner of the NFT
    function ownerOf(uint256 _tokenId) external view returns (address);

    /// @notice Transfers the ownership of an NFT from one address to
another address
    /// @dev Throws unless `msg.sender` is the current owner, an
authorized
    ///  operator, or the approved address for this NFT. Throws if
`_from` is
    ///  not the current owner. Throws if `_to` is the zero address.
Throws if
    ///  `_tokenId` is not a valid NFT. When transfer is complete, this
function
    ///  checks if `_to` is a smart contract (code size > 0). If so, it
calls
    ///  `onERC721Received` on `_to` and throws if the return value is
not
    ///  `bytes4(keccak256("onERC721Received(address,address,uint256,
bytes)"))`.
    /// @param _from The current owner of the NFT
```

```
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    /// @param data Additional data with no specified format, sent in
call to `_to`
    function safeTransferFrom(address _from, address _to, uint256
_tokenId, bytes data) external payable;

    /// @notice Transfers the ownership of an NFT from one address to
another address
    /// @dev This works identically to the other function with an extra
data parameter,
    ///  except this function just sets data to "".
    /// @param _from The current owner of the NFT
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    function safeTransferFrom(address _from, address _to, uint256
_tokenId) external payable;

    /// @notice Transfer ownership of an NFT -- THE CALLER IS
RESPONSIBLE
    ///  TO CONFIRM THAT `_to` IS CAPABLE OF RECEIVING NFTS OR ELSE
    ///  THEY MAY BE PERMANENTLY LOST
    /// @dev Throws unless `msg.sender` is the current owner, an
authorized
    ///  operator, or the approved address for this NFT. Throws if
`_from` is
    ///  not the current owner. Throws if `_to` is the zero address.
Throws if
    ///  `_tokenId` is not a valid NFT.
    /// @param _from The current owner of the NFT
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    function transferFrom(address _from, address _to, uint256 _tokenId)
external payable;

    /// @notice Change or reaffirm the approved address for an NFT
    /// @dev The zero address indicates there is no approved address.
    ///  Throws unless `msg.sender` is the current NFT owner, or an
authorized
    ///  operator of the current owner.
    /// @param _approved The new approved NFT controller
    /// @param _tokenId The NFT to approve
    function approve(address _approved, uint256 _tokenId) external
payable;

    /// @notice Enable or disable approval for a third party
("operator") to manage
    ///  all of `msg.sender`'s assets
    /// @dev Emits the ApprovalForAll event. The contract MUST allow
    ///  multiple operators per owner.
```

```
    /// @param _operator Address to add to the set of authorized
operators
    /// @param _approved True if the operator is approved, false to
revoke approval
    function setApprovalForAll(address _operator, bool _approved)
external;

    /// @notice Get the approved address for a single NFT
    /// @dev Throws if `_tokenId` is not a valid NFT.
    /// @param _tokenId The NFT to find the approved address for
    /// @return The approved address for this NFT, or the zero address
if there is none
    function getApproved(uint256 _tokenId) external view returns
(address);

    /// @notice Query if an address is an authorized operator for
another address
    /// @param _owner The address that owns the NFTs
    /// @param _operator The address that acts on behalf of the owner
    /// @return True if `_operator` is an approved operator for
`_owner`, false otherwise
    function isApprovedForAll(address _owner, address _operator)
external view returns (bool);
}

interface ERC165 {
    /// @notice Query if a contract implements an interface
    /// @param interfaceID The interface identifier, as specified in
ERC-165
    /// @dev Interface identification is specified in ERC-165. This
function
    ///  uses less than 30,000 gas.
    /// @return `true` if the contract implements `interfaceID` and
    ///  `interfaceID` is not 0xffffffff, `false` otherwise
    function supportsInterface(bytes4 interfaceID) external view
returns (bool);
}
```

A wallet/broker/auction application MUST implement the wallet interface if it will accept safe transfers.

```
/// @dev Note: the ERC-165 identifier for this interface is 0x150b7a02.
interface ERC721TokenReceiver {
    /// @notice Handle the receipt of an NFT
    /// @dev The ERC721 smart contract calls this function on the
recipient
    ///  after a `transfer`. This function MAY throw to revert and
reject the
    ///  transfer. Return of other than the magic value MUST result in
the
    ///  transaction being reverted.
    ///  Note: the contract address is always the message sender.
    /// @param _operator The address which called `safeTransferFrom`
function
    /// @param _from The address which previously owned the token
    /// @param _tokenId The NFT identifier which is being transferred
    /// @param _data Additional data with no specified format
    /// @return `bytes4(keccak256("onERC721Received(address,address,
uint256,bytes)"))`
    ///  unless throwing
    function onERC721Received(address _operator, address _from, uint256
_tokenId, bytes _data) external returns(bytes4);
}
```

The metadata extension is OPTIONAL for ERC-721 smart contracts (see "caveats", below). This allows your smart contract to be interrogated for its name and for details about the assets which your NFTs represent.

```
/// @title ERC-721 Non-Fungible Token Standard, optional metadata
extension
/// @dev See https://eips.ethereum.org/EIPS/eip-721
///  Note: the ERC-165 identifier for this interface is 0x5b5e139f.
interface ERC721Metadata /* is ERC721 */ {
    /// @notice A descriptive name for a collection of NFTs in this
contract
    function name() external view returns (string _name);

    /// @notice An abbreviated name for NFTs in this contract
    function symbol() external view returns (string _symbol);

    /// @notice A distinct Uniform Resource Identifier (URI) for a
given asset.
    /// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined
in RFC
    ///  3986. The URI may point to a JSON file that conforms to the
"ERC721
    ///  Metadata JSON Schema".
    function tokenURI(uint256 _tokenId) external view returns (string);
}
```

This is the "ERC721 Metadata JSON Schema" referenced above.

```
{
    "title": "Asset Metadata",
    "type": "object",
    "properties": {
        "name": {
            "type": "string",
            "description": "Identifies the asset to which this NFT
represents"
        },
        "description": {
            "type": "string",
            "description": "Describes the asset to which this NFT
represents"
        },
        "image": {
            "type": "string",
            "description": "A URI pointing to a resource with mime type
image/* representing the asset to which this NFT represents. Consider
making any images at a width between 320 and 1080 pixels and aspect
ratio between 1.91:1 and 4:5 inclusive."
        }
    }
}
```

The enumeration extension is OPTIONAL for ERC-721 smart contracts (see "caveats", below). This allows your contract to publish its full list of NFTs and make them discoverable.

```
/// @title ERC-721 Non-Fungible Token Standard, optional enumeration
extension
/// @dev See https://eips.ethereum.org/EIPS/eip-721
///  Note: the ERC-165 identifier for this interface is 0x780e9d63.
interface ERC721Enumerable /* is ERC721 */ {
    /// @notice Count NFTs tracked by this contract
    /// @return A count of valid NFTs tracked by this contract, where
each one of
    ///  them has an assigned and queryable owner not equal to the zero
address
    function totalSupply() external view returns (uint256);

    /// @notice Enumerate valid NFTs
    /// @dev Throws if `_index` >= `totalSupply()`.
    /// @param _index A counter less than `totalSupply()`
    /// @return The token identifier for the `_index`th NFT,
    ///  (sort order not specified)
    function tokenByIndex(uint256 _index) external view returns
(uint256);

    /// @notice Enumerate NFTs assigned to an owner
    /// @dev Throws if `_index` >= `balanceOf(_owner)` or if
    ///  `_owner` is the zero address, representing invalid NFTs.
    /// @param _owner An address where we are interested in NFTs owned
by them
    /// @param _index A counter less than `balanceOf(_owner)`
    /// @return The token identifier for the `_index`th NFT assigned to
`_owner`,
    ///   (sort order not specified)
    function tokenOfOwnerByIndex(address _owner, uint256 _index)
external view returns (uint256);
}
```

Caveats

The 0.4.20 Solidity interface grammar is not expressive enough to document the ERC-721 standard. A contract which complies with ERC-721 MUST also abide by the following:

- Solidity issue #3412: The above interfaces include explicit mutability guarantees for each function. Mutability guarantees are, in order weak to strong: `payable`, implicit nonpayable, `view`, and `pure`. Your implementation MUST meet the mutability guarantee in this interface and you MAY meet a stronger guarantee. For example, a `payable` function in this interface may be implemented as nonpayble (no state mutability specified) in your contract. We expect a later Solidity release will allow your stricter contract to inherit from this interface, but a workaround for version 0.4.20 is that you can edit this interface to add stricter mutability before inheriting from your contract.
- Solidity issue #3419: A contract that implements `ERC721Metadata` or `ERC721Enumerable` SHALL also implement `ERC721`. ERC-721 implements the requirements of interface ERC-165.
- Solidity issue #2330: If a function is shown in this specification as `external` then a contract will be compliant if it uses `public` visibility. As a workaround for version 0.4.20, you can edit this interface to switch to `public` before inheriting from your contract.
- Solidity issues #3494, #3544: Use of `this.*.selector` is marked as a warning by Solidity, a future version of Solidity will not mark this as an error.

If a newer version of Solidity allows the caveats to be expressed in code, then this EIP MAY be updated and the caveats removed, such will be equivalent to the original specification.

From Ethereum to Peerplays

In this section we will analyze ERC-721 standard, and describe how we can transfer its concepts to the Peerplays network.

```
/// @title ERC-721 Non-Fungible Token Standard
/// @dev See https://eips.ethereum.org/EIPS/eip-721
///  Note: the ERC-165 identifier for this interface is 0x80ac58cd.
interface ERC721 /* is ERC165 */ {
```

We can transfer concept of interface as an API, like we have database_api, network_api, etc in Peerplays. We can call it ntfs_api, and all APIs belonging to it will handle only NTFS.

```
    event Transfer(address indexed _from, address indexed _to, uint256
indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved,
uint256 indexed _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed
_operator, bool _approved);
```

Events, like those in Ethereum, do not exists in Peerplays/Graphene. Getting “events” from EOS (based on Graphene) is simulated by so called demux component, which is implemented as a JS script, running side by side with chain. So, we will not be able to support this concept out-of-the-box.

```
    /// @notice Count all NFTs assigned to an owner
    /// @dev NFTs assigned to the zero address are considered invalid,
and this
    ///  function throws for queries about the zero address.
    /// @param _owner An address for whom to query the balance
    /// @return The number of NFTs owned by `_owner`, possibly zero
    function balanceOf(address _owner) external view returns (uint256);
```

We will store the NFTs in multi_index, add index by_owner, filter elements by owner and count them.

```
    /// @notice Find the owner of an NFT
    /// @dev NFTs assigned to zero address are considered invalid, and
queries
    ///  about them do throw.
    /// @param _tokenId The identifier for an NFT
    /// @return The address of the owner of the NFT
    function ownerOf(uint256 _tokenId) external view returns (address);
```

We will store the NFTs in multi_index, add index by_id, find elment bz id, and return value of owner field.

```
    /// @notice Transfers the ownership of an NFT from one address to
another address
    /// @dev Throws unless `msg.sender` is the current owner, an
authorized
    ///  operator, or the approved address for this NFT. Throws if
`_from` is
    ///  not the current owner. Throws if `_to` is the zero address.
Throws if
    ///  `_tokenId` is not a valid NFT. When transfer is complete, this
function
    ///  checks if `_to` is a smart contract (code size > 0). If so, it
calls
    ///  `onERC721Received` on `_to` and throws if the return value is
not
    ///  `bytes4(keccak256("onERC721Received(address,address,uint256,
bytes)"))`.
    /// @param _from The current owner of the NFT
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    /// @param data Additional data with no specified format, sent in
call to `_to`
    function safeTransferFrom(address _from, address _to, uint256
_tokenId, bytes data) external payable;
```

All checks are applicable to Peerplays, except if _to address is smart contract, as it is not possible for smart contract to be object's owner in Peerplays. Event will not be raised, as events like those in Ethereum does not exists in Peerplays. _tokenId will be used to get the token from index by id. Function will have non-zero fee. In Peerplays, msg.sender is the account posting the transaction.

```
    /// @notice Transfers the ownership of an NFT from one address to
another address
    /// @dev This works identically to the other function with an extra
data parameter,
    ///  except this function just sets data to "".
    /// @param _from The current owner of the NFT
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    function safeTransferFrom(address _from, address _to, uint256
_tokenId) external payable;
```

Peerplays API does not support same functions with different signatures, but we can try to use default parameters here. If the data parameter is used, it will be as if we call first version of this function. If not, it will be as the data parameter is set to null.

```
    /// @notice Transfer ownership of an NFT -- THE CALLER IS
RESPONSIBLE
    ///  TO CONFIRM THAT `_to` IS CAPABLE OF RECEIVING NFTS OR ELSE
    ///  THEY MAY BE PERMANENTLY LOST
    /// @dev Throws unless `msg.sender` is the current owner, an
authorized
    ///  operator, or the approved address for this NFT. Throws if
`_from` is
    ///  not the current owner. Throws if `_to` is the zero address.
Throws if
    ///  `_tokenId` is not a valid NFT.
    /// @param _from The current owner of the NFT
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    function transferFrom(address _from, address _to, uint256 _tokenId)
external payable;
```

This function looks like previous one, with the exception of check whether the receiver is valid address. Eg in Ethereum, we can have smart contract, but in locked state, with no access to funds. If this contract becomes owner, it will not be able to do anything with the NFTs). I see no reason to skip this check. In Peerplays, we will just check whether the _to account id provided exists in account index. In Peerplays, msg.sender is the account posting the transaction.

```
    /// @notice Change or reaffirm the approved address for an NFT
    /// @dev The zero address indicates there is no approved address.
    ///  Throws unless `msg.sender` is the current NFT owner, or an
authorized
    ///  operator of the current owner.
    /// @param _approved The new approved NFT controller
    /// @param _tokenId The NFT to approve
    function approve(address _approved, uint256 _tokenId) external
payable;
```

Approved, in Ethereum, means that somebody else, who is not the owner, is able to manipulate some resource, eg funds, or in this case, NFTs. We can implement this feature in Peerplays with appropriate checks in operation evaluator. In Peerplays, msg.sender is the account posting the transaction.

```
    /// @notice Enable or disable approval for a third party
("operator") to manage
    ///  all of `msg.sender`'s assets
    /// @dev Emits the ApprovalForAll event. The contract MUST allow
    ///  multiple operators per owner.
    /// @param _operator Address to add to the set of authorized
operators
    /// @param _approved True if the operator is approved, false to
revoke approval
    function setApprovalForAll(address _operator, bool _approved)
external;
```

Same as previous function, but it will approve/disapprove handling of all NFTs from owner. We will not emit event. In Peerplays, msg.sender is the account posting the transaction.

```
    /// @notice Get the approved address for a single NFT
    /// @dev Throws if `_tokenId` is not a valid NFT.
    /// @param _tokenId The NFT to find the approved address for
    /// @return The approved address for this NFT, or the zero address
if there is none
    function getApproved(uint256 _tokenId) external view returns
(address);
```

This can be done by adding field approved_operators to the NFT object. This field will contain list of approved NFT operators. getApproved will return this list.

```
    /// @notice Query if an address is an authorized operator for
another address
    /// @param _owner The address that owns the NFTs
    /// @param _operator The address that acts on behalf of the owner
    /// @return True if `_operator` is an approved operator for
`_owner`, false otherwise
    function isApprovedForAll(address _owner, address _operator)
external view returns (bool);
```

We can implement this check by checking whether the _operator exists in approved_operator field in all NFTs owned by _owner.

```
interface ERC165 {
    /// @notice Query if a contract implements an interface
    /// @param interfaceID The interface identifier, as specified in
ERC-165
    /// @dev Interface identification is specified in ERC-165. This
function
    ///  uses less than 30,000 gas.
    /// @return `true` if the contract implements `interfaceID` and
    ///  `interfaceID` is not 0xffffffff, `false` otherwise
    function supportsInterface(bytes4 interfaceID) external view
returns (bool);
}

/// @dev Note: the ERC-165 identifier for this interface is 0x150b7a02.
interface ERC721TokenReceiver {
    /// @notice Handle the receipt of an NFT
    /// @dev The ERC721 smart contract calls this function on the
recipient
    ///  after a `transfer`. This function MAY throw to revert and
reject the
    ///  transfer. Return of other than the magic value MUST result in
the
    ///  transaction being reverted.
    ///  Note: the contract address is always the message sender.
    /// @param _operator The address which called `safeTransferFrom`
function
    /// @param _from The address which previously owned the token
    /// @param _tokenId The NFT identifier which is being transferred
    /// @param _data Additional data with no specified format
    /// @return `bytes4(keccak256("onERC721Received(address,address,
uint256,bytes)"))`
    ///  unless throwing
    function onERC721Received(address _operator, address _from, uint256
_tokenId, bytes _data) external returns(bytes4);
}
```

As Peerplays do not support events, we dont have to support these two interfaces.

```
/// @title ERC-721 Non-Fungible Token Standard, optional metadata
extension
/// @dev See https://eips.ethereum.org/EIPS/eip-721
///  Note: the ERC-165 identifier for this interface is 0x5b5e139f.
interface ERC721Metadata /* is ERC721 */ {
    /// @notice A descriptive name for a collection of NFTs in this
contract
    function name() external view returns (string _name);

    /// @notice An abbreviated name for NFTs in this contract
    function symbol() external view returns (string _symbol);

    /// @notice A distinct Uniform Resource Identifier (URI) for a
given asset.
    /// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined
in RFC
    ///  3986. The URI may point to a JSON file that conforms to the
"ERC721
    ///  Metadata JSON Schema".
    function tokenURI(uint256 _tokenId) external view returns (string);
}
```

```
{
    "title": "Asset Metadata",
    "type": "object",
    "properties": {
        "name": {
            "type": "string",
            "description": "Identifies the asset to which this NFT
represents"
        },
        "description": {
            "type": "string",
            "description": "Describes the asset to which this NFT
represents"
        },
        "image": {
            "type": "string",
            "description": "A URI pointing to a resource with mime type
image/* representing the asset to which this NFT represents. Consider
making any images at a width between 320 and 1080 pixels and aspect
ratio between 1.91:1 and 4:5 inclusive."
        }
    }
}
```

Supporting ERC721Metadata interface will not be a problem, as we routinely use json for data storage and object serialization.

```
/// @title ERC-721 Non-Fungible Token Standard, optional enumeration
extension
/// @dev See https://eips.ethereum.org/EIPS/eip-721
///  Note: the ERC-165 identifier for this interface is 0x780e9d63.
interface ERC721Enumerable /* is ERC721 */ {
    /// @notice Count NFTs tracked by this contract
    /// @return A count of valid NFTs tracked by this contract, where
each one of
    ///  them has an assigned and queryable owner not equal to the zero
address
    function totalSupply() external view returns (uint256);

    /// @notice Enumerate valid NFTs
    /// @dev Throws if `_index` >= `totalSupply()`.
    /// @param _index A counter less than `totalSupply()`
    /// @return The token identifier for the `_index`th NFT,
    ///  (sort order not specified)
    function tokenByIndex(uint256 _index) external view returns
(uint256);

    /// @notice Enumerate NFTs assigned to an owner
    /// @dev Throws if `_index` >= `balanceOf(_owner)` or if
    ///  `_owner` is the zero address, representing invalid NFTs.
    /// @param _owner An address where we are interested in NFTs owned
by them
    /// @param _index A counter less than `balanceOf(_owner)`
    /// @return The token identifier for the `_index`th NFT assigned to
`_owner`,
    ///   (sort order not specified)
    function tokenOfOwnerByIndex(address _owner, uint256 _index)
external view returns (uint256);
}
```

All function from ERC721Enumerable interface may be implemented by using appropriate multi_index properties and functions.