D/BOND

EIP-3475

Abstract Storage Bonds

Yu LIU
Email: yu@debond.org
Twitter: @DebondProtocol

ERC-3475

Interface for **tokenized obligations** with **abstract on-chain metadata storage**

1.2 Token standards

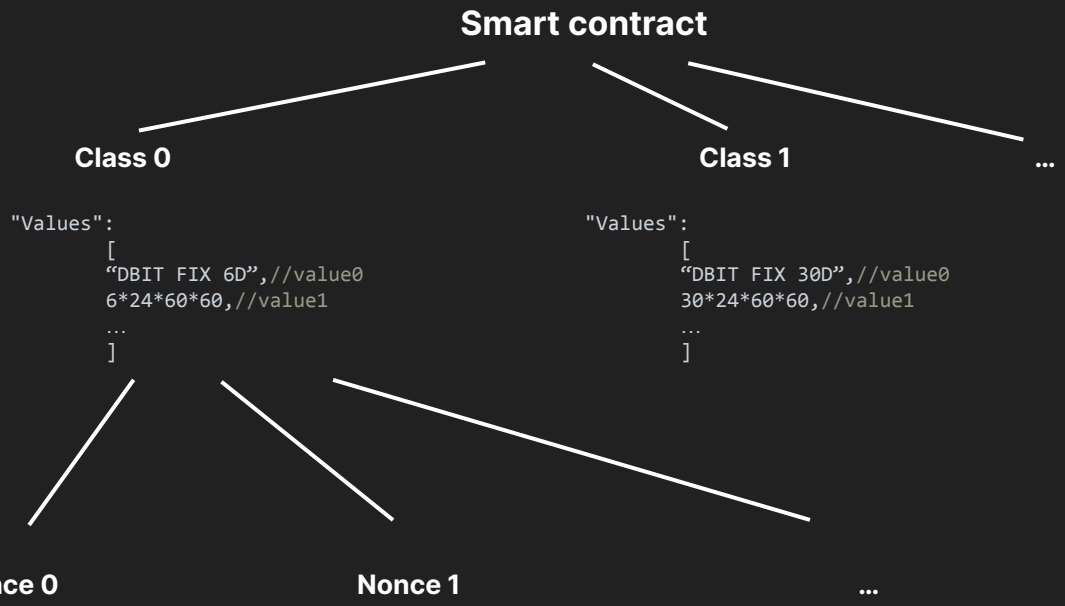
*Both fungible and non-fungible

	ERC-20	ERC-721	ERC-1155	ERC-3525	**ERC-3475**
Fungible	Yes	No	semi-fungible*	semi-fungible*	semi-fungible*
Metadata	N/A	off-chain	off-chain	off-chain	on-chain
Token Id	N/A	single	single	multidimensional	multidimensional
Batch TX	N/A	N/A	arrays	N/A	tuple[]
Storage	specified	specified	specified	specified	Adaptable
Types of supply	1	N/A	1	1	3

1.3 Who uses ERC-3475 token standard?

- **Traditional financial institutions**
  - Using ERC-3475, investment banks can convert their financial product into digital asset, facilitating the exchange and the openness of the market.
- **DeFi project**
  - The Multi layered pool provides more efficient AMM solution.
  - Securitise the LP token, and create a new market for securities.
- **Other DApp**
  - GameFi or other DApp, have now a better solution to store and manipulate on-chain data efficiently.
  - Because of the on-chain metadata itself is abstract, ERC-3475 can be used in other industries.

1.4 Smart Contract structure



Class metadata keys and description

```
[
        {
        "title": "symbol", //key0
        "_type": "string",
        "description": "Lorem ipsum..."
        },
        {
        "title": "period", //key1
        "_type": "uint",
        "description": "Lorem ipsum..."
        },
        ...
]
```

Nonce metadata keys and description

```
 [
         {
         "title": "maturity", //key0
         "_type": "uint",
         "description": "Lorem ipsum..."
         },
         ...
 ]
```

D/BOND

https://debond.org

1.5.1 Class, nonce Id and metadata structure

IERC3475.sol

```
struct Values{
        string stringValue;
        uint uintValue;
        address addressValue;
        bool boolValue;
    }
struct Metadata{
        string title;
        string _type;
        string description;
        }
```

ERC3475.sol

```
struct Nonce {
        mapping(uint256 => IERC3475.Values) _values;
        mapping(address => uint256) balances;
        mapping(address => mapping(address => uint256)) allowances;
        address owner;
        uint256 _activeSupply;
        uint256 _burnedSupply;
        uint256 _redeemedSupply;
}
struct Class {
        mapping(uint256 => IERC3475.Values) _values;
        mapping(uint256 => IERC3475.Metadata) _nonceMetadata;
        mapping(uint256 => Nonce) nonces;
}
mapping(uint256 => IERC3475.Metadata) _classMetadata;
```

1.5.2 Metadata for front-end

JSON

```
[
        {
        "title": "symbol",
        "_type": "string",
        "description": "Lorem ipsum..."
        "values": ["Lorem", "Lorem", "Lorem",...],
        },
        {
        "title": "issuer",
        "_type": "string",
        "description": "Lorem ipsum..."
        "values": ["Lorem", "Lorem", "Lorem",...],
        },
        {
        "title": "token_address",
        "_type": "address",
        "description": "Lorem ipsum..."
        "values":["0x4...", "0x4...", "0x4...",...]
        },
        {
        "title": "period",
        "_type": "uint",
        "description": "Lorem ipsum..."
        "values": [0, 0, 0,...]
        },
        ...
]
```

Script

Search for keys, and find the values

Front end

D/BOND

https://debond.org

1.6 Access control

Use class metadata to define roles

```
[
        {
        "title": "issuer_address", //role1
        "_type": "address",
        "description": "Lorem ipsum..."
        },
        ...
        {
        "title": "operator_address", //role2
        "_type": "address",
        "description": "Lorem ipsum..."
        },
        ...
]
```

Import and use "@openzeppelin's access control to create modifier

```
struct RoleData {
      mapping(address => bool) members;
      bytes32 adminRole;
  }
mapping(bytes32 => RoleData) private _roles;
private _roles["issuer"] = classes[0]._values[5].addressValue;
modifier onlyRole(bytes32 role) {
      _checkRole(role);
      _;
  }
```

Require the role for certain functions

```
function issue(address to, Transaction[] calldata transactions) external onlyRole("issuer");
```

1.7 Batch transactions

```
struct TRANSACTION {
        uint256 classId;
        uint256 nonceId;
        uint256 _amount;
}
/**
* @dev allows the transfer of a bond from an address to another (either single or in batches).
* @param _from argument is the address of the holder whose balance about to decrease.
* @param _to argument is the address of the recipient whose balance is about to increased.
*/
function transferFrom(address _from, address _to, TRANSACTION[] calldata _transaction) external;
/**
* @dev allows the transfer of allowance from an address to another (either single or in batches).
* @param _from argument is the address of the holder whose balance about to decrease.
* @param _to argument is the address of the recipient whose balance is about to increased.
*/
function transferAllowanceFrom(address _from, address _to, TRANSACTION[] calldata _transaction) external;
```

1.7 Batch transactions

```
struct TRANSACTION {
        uint256 classId;
        uint256 nonceId;
        uint256 _amount;
}
/**
* @dev allows the transfer of a bond from an address to another (either single or in batches).
* @param _from argument is the address of the holder whose balance about to decrease.
* @param _to argument is the address of the recipient whose balance is about to increased.
*/
function transferFrom(address _from, address _to, TRANSACTION[] calldata _transaction) external;
/**
* @dev allows the transfer of allowance from an address to another (either single or in batches).
* @param _from argument is the address of the holder whose balance about to decrease.
* @param _to argument is the address of the recipient whose balance is about to increased.
*/
function transferAllowanceFrom(address _from, address _to, TRANSACTION[] calldata _transaction) external;
```

1.8 Types of supply

```
/**
* @dev Returns the total supply of the bond in question.(totalSupply = redeemedSupply + activeSupply + burnedSupply)
*/
function totalSupply(uint256 classId, uint256 nonceId) external view returns (uint256);
/**
* @dev Returns the redeemed supply of the bond in question.
*/
function redeemedSupply(uint256 classId, uint256 nonceId) external view returns (uint256);
/**
* @dev Returns the active supply of the bond in question.
*/
function activeSupply(uint256 classId, uint256 nonceId) external view returns (uint256);
/**
* @dev Returns the burned supply of the bond in question.
*/
function burnedSupply(uint256 classId, uint256 nonceId) external view returns (uint256);
```

1.9 Events

```
/**
* @notice MUST trigger when tokens are transferred, including zero value transfers.
*/
event Transfer(address indexed _operator, address indexed _from, address indexed _to, TRANSACTION[] _transaction);
/**
* @notice MUST trigger when tokens are issued
*/
event Issue(address indexed _operator, address indexed _to, TRANSACTION[] _transaction);
/**
* @notice MUST trigger when tokens are redeemed
*/
event Redeem(address indexed _operator, address indexed _from, TRANSACTION[] _transaction);
/**
* @notice MUST trigger when tokens are burned
*/
event Burn(address indexed _operator, address indexed _from, TRANSACTION[] _transaction);
/**
* @dev MUST emit when approval for a second party/operator address to manage all bonds from a classId given
for an owner address is enabled or disabled (absence of an event assumes disabled).
*/
event ApprovalFor(address indexed _owner, address indexed _operator, bool _approved);
```

2.1 Use case -Bonds and financial derivatives

```
// define "period of the bond class";
_classMetadata[5].title = "period";
_classMetadata[5]._type = "uint";
_classMetadata[5].description = "details about issuance and redemption time";
classes[0]._values[5].uintValue = 180 days;

// define "maturity of the nonce";
classes[0]._nonceMetadata[0].title = "maturity";
classes[0]._nonceMetadata[0].title = "uint";
classes[0]._nonceMetadata[0].description = "maturity date";

// write the time of maturity to nonce values, in other implementation, a create nonce function can be added
classes[0].nonces[0]._values[0].uintValue = block.timestamp + Classes[0]._values[1].uintValue;
classes[0].nonces[1]._values[0].uintValue = block.timestamp + Classes[0]._values[1].uintValue + 1 days;
classes[0].nonces[2]._values[0].uintValue = block.timestamp + Classes[0]._values[1].uintValue + 2 days;
```

2.2 Use case -Multi-Layer Pool

TOKENS

wETH

wBTC

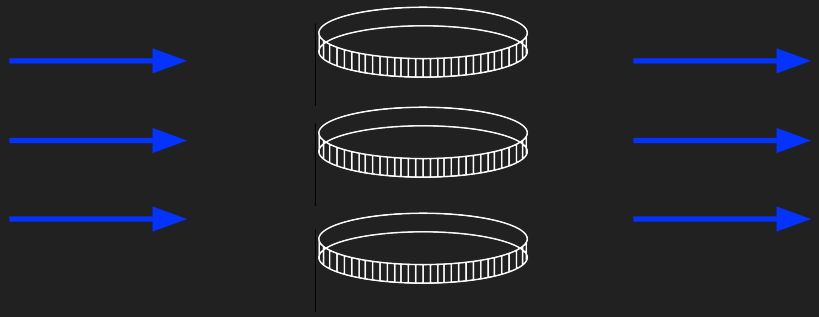
BNB

DOT

USDT

...

Multi-Layer Pool (single smart contract)



ERC 3475 LP token

Class 0. Fixed-rate 6M wETH Bond
(07-19-2022, nonce 119)

Class 1. Fixed-rate 3M BNB Bond
(07-22-2022, nonce 122)

Class 6. Floating-rate 6M BNB Bond
(07-19-2022, nonce 119)

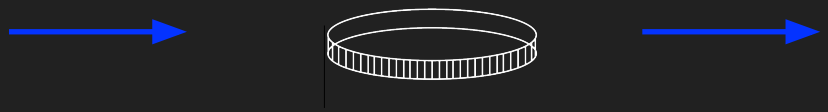
Class 6. Floating-rate 6M BNB Bond
(07-22-2022, nonce 122)

...

TOKENS

wETH
USDT

Virtual pair (AMM)



Prove of deposit

Class 0. Fixed-rate 6M wETH Bond
(07-19-2022, nonce 119)

3.1 What can EIP-3475 bring to Ethereum?

- **On-chain metadata storage:**
  - On-chain metadata can both be read efficiently by the frontend and the smart contract.
  - Allow the conception of more sophisticated smart contract logic.
- **Decentralised bond and derivatives**
  - Standardize on-chain financial product such as bond and derivatives.
- **Efficiencies in AMM use case**
  - Reduce 85% gas fee in creating pair, 32% in adding liquidity.
  - Reduce the slippage when swapping.
- **Abstract and adaptable storage:**
  - The values and their functions are not specified in the EIP, making them adaptable.
  - The complex data structure and its adaptability allow it to be used efficiently in any use case.

3.2 Why standardize?

- **What current token standards can’t offer**
  - We need a more adaptive token standard to represent obligations and other complex financial product.

- **Front-end and applications**
  - Using a standardized interface, we can read more precise and detailed data from the smart contract’s storage.
  - A set of front end application can be used in most of the use cases, facilitate the accessibility and explicitly of the Ethereum.

3.3 Other considerations

- **Class and nonce structure.**
  - Classes and nonces built with array instead of mapping.
  - Define symbol, logo, or maturity value in the contract, using separated variables.
- **Storage**
  - Using TLV method (tag, length, value), get everything stored in byte.
  - Convert string stored in metadata to address, uint and bool.
- **Metadata for the front end**
  - Define the JSON metadata structure in the EIP.

3.4 Benchmarks

Gas consumed

	Curve.fi	Uniswap V3	Multi-layer pool
Swap	132,556	138,012	135,583
Create pair	1,073,216	4,557,546	151,383
Add liquidity	811,801	216,765	147,780

Discussion:
ethereum-magicians

Contact:
info@debond.org