Time Bound Token

Ricsson Ngo

@Timelord.Eth

ricsson@timeswap.io

Shatabarto Bhattacharya

@riksucks

shatabarto@timeswap.io

Abstract

Time Bound Token or TBT is a token that makes it possible to transfer and manage ownership of the timeline of an asset, essentially a pure representation of an option. It can represent time bounded assets like rentable NFT, escrow positions, time restricted governance, generalized options, renting real estates, and many more. TBT works by cutting tokens (could be fungible or non-fungible) into periodic timelines, which can be owned by different addresses.

[

TBT Diagram

1840×244 22.6 KB

](https://ethresear.ch/uploads/default/original/2X/8/88df7c5cdf6d7bd826d3d47d73c8ccd8fe4de725.png)

Figure 1: 100 Time Bound Token A with monthly period

Introduction

TBT has been designed keeping utility in mind, hence TBT can be termed as a utility token. The owner of the TBT has the right to utilize the underlying asset for some economic or financial benefit via calling the utility function. Ownership of the token is determined by checking if present time is part of the claim of the timeline by the user.

Example of these benefits could be the following:

- Minting new Crypto Kitty eggs with the wrapped Crypto Kitty.
- Collecting fees with the wrapped Uniswap liquidity position.
- Calling a governance related function.
- Swapping between USDC and ETH following a constant sum formula (options).
- Representing renting of real estates.
- Recurring Payments

Let us peruse the example above for a thought experiment as an example use case. Suppose, Alice owns 100 TBT from month 0 to month 1 and Bob owns 100 TBT from month 1 to month 2. Now let's assume that 0.5 months have passed from month 0, then Alice can call the utility function as many times as she wants, while Bob cannot, as the present time is a part of the claim of timeline by Alice. Alice will lose the ability to call the utility function after month 1, while Bob must wait till month 1 to be able to benefit from the utility function. TBT simply expires and disappears when its full time period passes.

TBT can be transferred like a normal token. It should be noted that transferring here represents transfer of claim and the amount of tokens. For example Alice can transfer 50 TBT from month 0 to month 1 to Bob, thus Alice will have 50 TBT from month 0 to month 1 remaining, while Bob will then have 50 TBT from month 0 to month 1 and 100 TBT from month 1 to month 2. Charlie can transfer 100 TBT from month 6 to infinity to Oscar, thus Charlie will have 100 TBT from month 2 to month 6 remaining, while Oscar will then have 100 TBT from month 6 to infinity.

Implementation Details

Here are some early interface draft implementations of TBT. We plan to further develop the implementation. Feedbacks and discussions are welcomed. We plan to turn this into an EIP.

We will follow a multi token standard similar to ERC1155, where we have ids that represents different tokens. A token

supply of one means it is an NFT, while token supply of more than one represents fungible tokens.

- function period(uint256 id) external view returns (uint256 seconds)

This function returns the number of seconds of one period of a TBT with the given id. We call the timestamp between two periods as time ticks. Generally, we do not want the period to be too short, as this will lead to too high of gas cost in the TBT. We also do not want the period to be too long, as we lose too much flexibility of the TBT. Some general social standards are weekly, monthly, quarterly, semi-annual, and annual.

- function shift(uint256 id) external view returns (uint256 seconds)

This function returns the number of initial shifts of seconds, where we start counting the period. The shift amount should be smaller than the period length.

- function transfer(address to, uint256 id, uint64 start, uint64 end, uint256 amount, bytes calldata data) external

This function lets the owner transfer TBT to a target address given the start and end of the period of the TBT. The start and end must be divisible by the period after subtracting the shift. If the start is smaller than the current block timestamp, it will default to the start time tick of the current period. If the end is zero, we assume it to be infinity. The data structure should be implemented with a linked mapping for optimal gas efficiency and minimal updates.

- function balanceOf(address owner, uint256 id) external view returns (bytes memory balance)

This function will return bytes that represent the whole timeline position of an owner. It will require the caller to decode it with a pure function as shown below:

function decodeBalance(bytes memory balance) internal pure returns (TimeDelta[] memory timeDeltas)

struct TimeDelta { uint64 time; int192 delta; }

An array of time deltas is an efficient data structure to represent timeline positions. The time field represents the time tick where there is a delta change position for the owner. The delta field represents the positive or negative change of TBT after the time tick.

For example supposed we get time deltas of the following:

- { time: 1,700,000,000, delta: 100 }
- { time: 1,700,010,000, delta: -50 }
- { time: 1,700,030,000, delta: 150 }

This means that the owner has 100 TBT from block timestamp 1,700,000,000 to 1,700,010,000. Then 100 – 50 = 50 TBT from block timestamp 1,700,010,000 to 1,700,030,000. And finally, 100 – 50 + 150 = 200 TBT from block timestamp 1,700,030,000 to infinity.

Do note that such implementation, where we use bytes and decoder, is only required for returning future claims up to any time. We might change the implementation to something more practical where we cap the return to a max limit.

Note that the sum of time delta of each time tick must not be negative. Therefore minting, burning, and transferring tokens must guarantee this behavior. Also, as time moves forward, the balance of the owner must only show the latest time tick of the current time period onwards.

For example, using the same example as above, suppose the block timestamp has turned 1,700,010,010. The balanceOf should show these time deltas:

- { time: 1,700,010,000, delta: 50 }
- { time: 1,700,030,000, delta: 150 }

There are other functions not yet shown in this document, like multi transfer, update, paginated balanceOf, paginated update, metadata, onReceived, etc. The functions above are the key difference of TBT from current token standards.

Another potential implementation of TBT is having one single repository of TBT in one contract, where the basic implementation of updating timeline balance, minting, burning, and transfer is implemented. Anyone can initialize a TBT for a given id. For extensibility, we have callbacks (hooks) at multiple points of the TBT cycle. This has the benefit of potentially not requiring approve and transferFrom, as contracts interfacing with TBT can utilize the multi token transfer with data.

Future Scope

With the time bound tokenization, we can now have a standard to creatively financialize time bounded assets. We can create AMM for these assets, for example a market that swaps between TBT from month 0 to month 1 and TBT from month

1 to infinity. Example of current protocols with similar functionality are the following:

- Pendle Finance that cuts an existing yield bearing assets into two timelines.
- The yield token, which is from present to maturity, where the utility function collects the yield gains before maturity.
- The principal token, which is from maturity onwards, where users can unwrap and get back the principals after maturity.
- The protocol lets users swap between these two positions for fixed yields and discounts.
- The yield token, which is from present to maturity, where the utility function collects the yield gains before maturity.
- The principal token, which is from maturity onwards, where users can unwrap and get back the principals after maturity.
- The protocol lets users swap between these two positions for fixed yields and discounts.
- Timeswap that swaps tokens through time, which also cuts the assets into two timelines.
- Swap present tokens for future tokens, where the protocol swaps tokens from before maturity for tokens from maturity onwards, which lets users purchase discounted tokens.
- Swap future tokens for present tokens, where the protocol swaps tokens from the maturity onwards for tokens from before maturity, which lets users leverage up with no liquidation.
- Swap present tokens for future tokens, where the protocol swaps tokens from before maturity for tokens from maturity onwards, which lets users purchase discounted tokens.
- Swap future tokens for present tokens, where the protocol swaps tokens from the maturity onwards for tokens from before maturity, which lets users leverage up with no liquidation.

TBT will expand the functionality of future iteration of these protocols as well as attract new designs in the fixed maturity financial space. This token standard could be the spark that expands the DeFi space to the fixed maturity space, which has a large untapped market from traditional finance.

Renting NFTs for gaming is also a popular functionality. Game guilds would rent out their NFTs for players to earn game rewards. Having TBT will expand the flexibility and functionality of renting out NFTs.