Tinyman Governance Protocol Specification

Context

The TINY governance token will be launched in the next months. The core utility for token holders is participating in decision making for the Tinyman Protocol. In the months following the token launch all of the Tinyman related accounts controlled by the development team will be rekeyed to contract accounts controlled entirely by governance processes. This includes the fee manager & fee collector accounts of the AMM, the TINY treasury, the NFT creator account and the accounts that receive TDR rewards from Algorand Foundation.
The current off-chain farming system will also move to an on-chain system with reward distribution controlled by TINY voting.

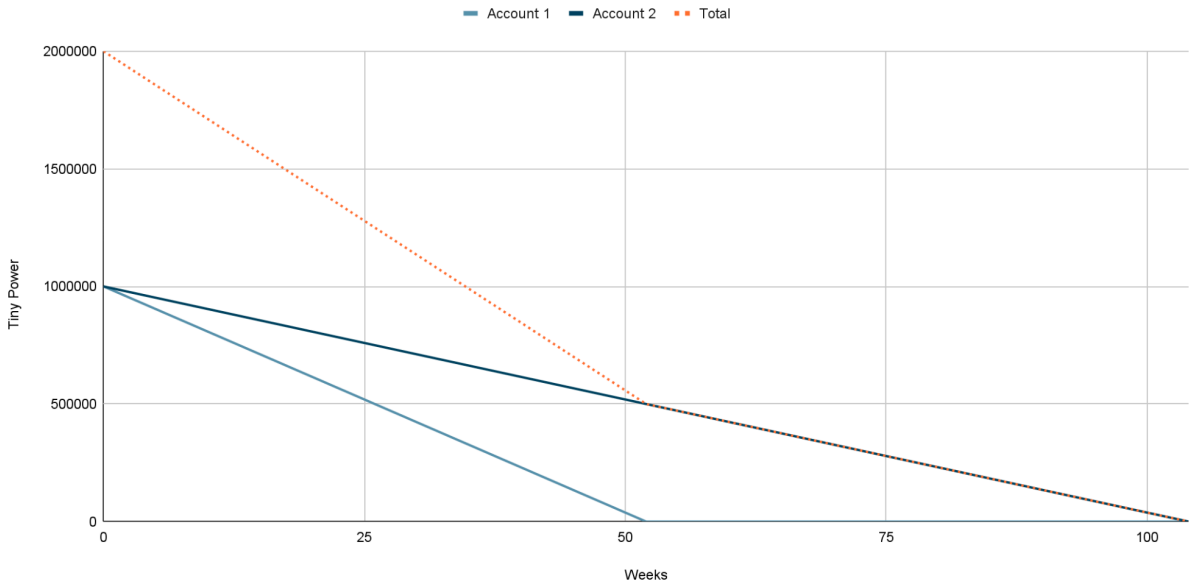
The transition of these accounts and processes will be gradual over a number of months. After the initial launch the governance app will allow for arbitrary proposals and on-chain voting by TINY holders but proposals will initially be filtered by the team as necessary. Additionally the treasury/management accounts will not immediately be rekeyed to contract accounts so the development team will be responsible for executing the decisions of governance.

After the trial period the team will rekey the accounts to contracts to allow for permissionless execution of accepted governance proposals. Eventually the team will transfer all managerial functionality of the voting contracts to governance itself.

The Vault app which will hold locked TINY will be immutable from launch. The voting and rewards apps will be mutable to allow for potential fixes during the trial period and functionality upgrades later. The development team will initially have control over such updates but later this control will be transferred to governance controlled contracts.

Overview

Being a Tinyman Governor requires the locking of TINY tokens for a certain period, similar to the Curve DAO. Accounts will acquire Tiny Power when they lock TINY. The amount of Tiny Power increases with the quantity of TINY locked or the duration of the lock. Tiny Power is calculated dynamically and decreases linearly over time. For example, initially, an account that locks 2M TINY for 2 years will have the same Tiny Power as one that locks 4M TINY for 1 year.



While implementing Tinyman Governance Contracts, Curve and OpenZeppelin (GovernorAlpha and Bravo) were used as reference implementations.

[Curve DAO Documentation]
[Curve Voting Escrow Contract]
[OpenZeppelin Documentation]
[OpenZeppelin Governance Contracts]

TINY ASA (Algorand Standard Asset)

The TINY token will be created by the Tinyman team, and it will have 1 billion whole units with 6 decimals, making the total amount 1,000,000,000,000,000 (15 zeros). All parameters representing TINY or Tiny Power are in micro-units. For example, to set a 10 TINY minimum lock limit in the governance contract, 10,000,000 (7 zeros) must be used.

Asset Parameters

1. Total Amount: 1,000,000,000,000,000
2. Decimal: 6
3. Creator: RIKLQ5HEVXAOAWYSW2LGQFYGWVO4J6LIAQQ72ZRULHZ4KS5NRPCCKYPCUU
4. Unit Name: TINY
5. Asset Name: TINY
6. URL: https://tinyman.org
7. Manager: -
8. Reserve: RIKLQ5HEVXAOAWYSW2LGQFYGWVO4J6LIAQQ72ZRULHZ4KS5NRPCCKYPCUU
9. Freeze: -
10. Clawback: -
11. Default Frozen: False
12. Meta Data Hash: -

Algorand Asset Parameters

Applications (Contracts)

Tinyman Governance comprises four Algorand applications (smart contracts): Vault, Rewards, Proposal Voting, and Staking Voting.

The Vault App contains the core of the governance mechanism. It manages the locking process and contains the Tiny Power (voting weight) calculation. Serving as the source of truth, other contracts can access Tiny Power data and its derivatives on-chain using inner transactions.

The Rewards App distributes “Governance Rewards” (TINY) to the community. These rewards will be allocated to Tinyman Governors each week, based on the TINY Power they hold.

Thanks to the Staking Voting App, governors will determine the distribution of “Liquidity Mining Rewards” (TINY) among the pools. TINY rewards will be distributed to Tinyman's liquidity miners through monthly farming programs. Each month, governors can allocate their Tiny Power to different liquidity pools (V2), with TINY rewards being distributed to the top N pools based on the votes

The Proposal Voting App manages the on-chain voting for governance proposals. It enables eligible governors to create proposals, and allows all governors to cast votes to express their opinions on these proposals.

Common Solutions

Storage

The applications utilize both global and box storage. Global storage, which has very limited space, is used for storing contract settings. For unlimited storage needs, such as governor and proposal data, box storage is preferred. Governors cover the cost of boxes with an additional transaction in their governance transaction groups.

Array Implementation

The Algorand Virtual Machine (AVM) does not have a built-in array data type. Consequently, we have implemented an array of structs. Our approach involves the creation of a dynamic 2D array-like data structure using boxes. To read or write data, the first step is to calculate the box index to determine which box to access. Next, the array index is calculated to identify the starting byte index within the box for reading or writing operations. When the boxes reach full capacity, a new box is created to expand the available space.

The index calculation formula is: $Index = Box\ Index * Box\ Array\ Length + Array\ Index.$ $Ex: 6 = 1 * 4 + 2, if\ box\ array\ length\ is\ 4$

	Array Index 0	Array Index 1	Array Index 2	Array Index 3
Box 0	X (Index 0)	X (Index 1)	X (Index 2)	X (Index 3)
Box 1	X (Index 4)	X (Index 5)	- (Index 6)	- (Index 7)

Box storage provides more space but it has its own limits. Read budget and size limit were considered while implementing arrays. The maximum box size is 32KB and a box reference provides a 1KB read budget to the application. An application call can have a total 8 references including boxes, accounts, foreign applications, foreign assets. We chose to set box sizes less than or equal to 1KB, so they work with 1 reference.

Budget Increase

In Algorand, the operational capacity of an application is constrained and is referred to as the opcode budget. This budget is shared across group transactions. If the operations of an application exceed the available opcode budget, additional application transactions can be added to the same transaction group to increase this budget. These supplemental transactions can be either outer or inner transactions.

All governance applications feature an "increase_budget" method to boost the opcode budget as required. This method is designed to be without side effects and enables the successful creation of an application call transaction with minimal opcode use.

Parallel to the opcode budget, boxes also have read and write budgets. Each box reference contributes a 1KB budget, which, like the opcode budget, is shared across group transactions.

The "increase_budget" method serves to augment both the opcode and the box read/write budgets. Notably, the "increase_budget" methods in the Rewards, Proposal Voting, and Staking Voting applications can also initiate inner transactions to the Vault App. This is done to increase the opcode budget using inner transactions, thus not exhausting the maximum limit of group transactions.

Vault App:
App call:
- Sender: User Address
- Index: Vault App ID
- OnComplete: NoOp
- App Args: [“increase_budget”]
- Fee: min_fee

Other Apps:
App call:
- Sender: User Address
- Index: Staking Voting, Proposal Voting, Rewards App ID
- OnComplete: NoOp
- App Args: [“increase_budget”, <int: inner increase budget txn count>]
- Foreign Apps: [Vault App ID] (Optional: Required for inner transactions)
- Fee: (inner txn count + 1) * fee

Proposal ID

Proposals should include immutable data encompassing all necessary information, such as title, description, aim, and actions, which would be shared with governors. This enables governors to analyze the proposal thoroughly before casting their votes. Commonly, in decentralized systems, data is uploaded using IPFS (InterPlanetary File System), resulting in the proposal ID being the CID (Content Identifier) of the metadata uploaded to IPFS.

However, we have decided not to use IPFS for the sake of simplicity. Instead, the raw metadata will be uploaded to the Tinyman Analytics Backend. To ensure its immutability, governors can

independently calculate the CID using the metadata and then compare it with the proposal ID on-chain.

Technically, the proposal ID will be in the CID format in the V1 version, utilizing base 32 encoding, the raw codec, and the sha2-256 hash function.

Get Box

On Algorand, on-chain applications (contracts) are only able to read their own boxes and cannot access the boxes of other applications. However, data sharing between applications is possible by reading the last log emitted by another application.

Applications such as the Vault and Proposal Voting have methods like “get_tiny_power_of”, “get_proposal_state”, and “has_voted”, which facilitate interactions with other applications built upon them. To provide flexibility for future requirements, the “get_box_method” shares the raw box data with other applications. This allows them to process the data according to their specific needs. The method returns the existence status of the box and its data.

Transaction Group:

1. App call:
   Sender: User Address
   Index: App ID
   OnComplete: NoOp
   App Args: [“get_box”, <box_name>]

Logs:

- return: uint16,byte[] (length, box_data)

Logs

Logs, created using the log opcode, are frequently utilized by governance applications. One of their primary functions is to facilitate data sharing between applications. While applications cannot access the box storage of another application directly, they can read the last log emitted by other applications using the LastLog opcode. To retrieve data, applications issue an inner transaction.

Another important use of logs is to provide event logs for off-chain trackers. These trackers can then parse events based on predefined formats or criteria. Unlike global and local storage deltas, changes made within box storage are not automatically added to block data. This is why logs are particularly important for off-chain block followers, as they offer a means to track and understand changes and events that are not directly reflected in the block data.

The contracts follow the ARC-28 Event Log Spec but do not adhere to the ARC-4 ABI Spec.

Vault App

TINY holders can lock their assets for a chosen duration to obtain Tiny Power (voting weight) and to earn rewards. The Vault App oversees this locking process and ensures the release of the TINY assets back to the governor once the predetermined lock end time is reached. Additionally, this app handles the calculations related to Tiny Power and provides the necessary data to other applications. Conceptually, this contract draws inspiration from Curve's Voting Escrow CRV (veCRV) model.

Similarities:

- Tiny Power is equivalent to veCRV in this contract. Accounts need to lock their governance tokens to gain voting weight (Tiny Power), which decreases linearly over time. The locked amount and remaining lock duration are parameters of power calculation.
- The maximum lock duration is approximately 4 years (1460 days). Locking X TINY for 4 years initially grants an equivalent amount of X Tiny Power.
- Lock end timestamps are predetermined, and funds are released weekly at consistent times. These timestamps must be multiples of 604800 seconds (7 days).
- Voting weight must be recorded historically and be retrievable for any given past timestamp.

Differences:

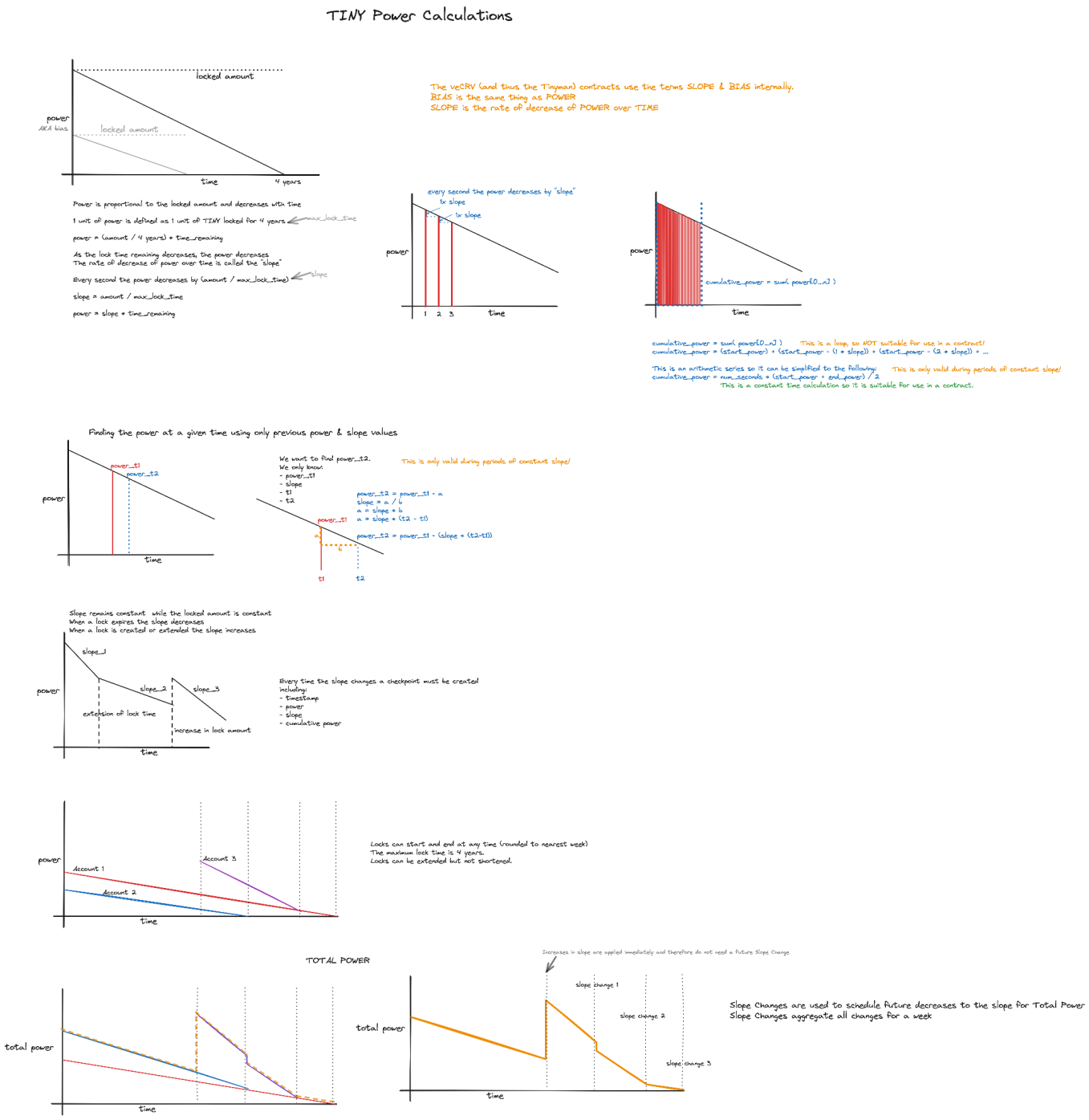
- A minimum amount requirement is introduced for creating locks and increasing lock amounts.
- A minimum lock duration requirement is implemented for creating locks and extending lock durations.
- The concept of Cumulative Power is introduced for the rewards mechanism. Account rewards are proportional to their voting power, calculated using their cumulative power delta and the total cumulative delta between reward periods, as utilized by the Rewards App.
- AVM and EVM differences include the lack of built-in array and hashmap types in AVM. There are also differences in on-chain data sharing, external views, and inner transactions. Storage (box) references must be calculated off-chain during transaction preparation to access the data.

Requirements:

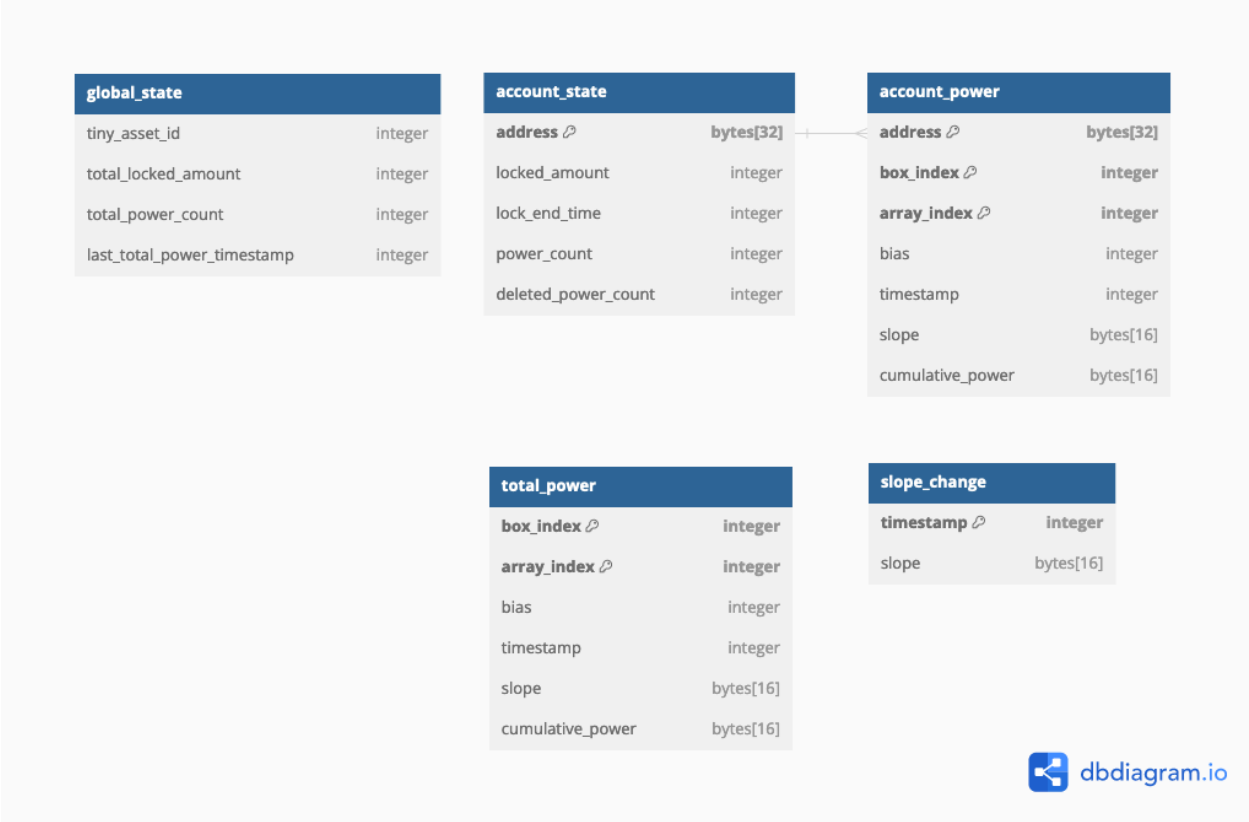
- Users can vote for each proposal using their voting power at the time of proposal creation. Therefore, the app must store historical voting power per account and in total, ensuring accessibility at any given time on-chain.
- Lock end timestamps will be multiples of 604800 seconds (Thursday 00:00:00 UTC), grouping lock end times and scheduled slope changes. This grouping reduces storage costs and facilitates slope change applications.

- Governors will start accumulating rewards immediately and can claim them at the end of each week (midnight UTC between Wednesday and Thursday). Releasing rewards requires a permissionless transaction (application call).
- Lock end times must be set in the future and be more than 4 weeks from the start time or the latest block time.
- Maximum lock time is 126144000 seconds (approximately 4 years).
- Minimum lock amount is 10 TINY (10,000,000).
- Minimum lock increase amount is 10 TINY (10,000,000).
- Minimum lock end time increment is 4 weeks.

Tiny Power Calculations



Storage Overview



https://dbdiagram.io/d/Vault-App-649593d602bd1c4a5ef796d9

Global State

- Tiny Asset ID
- Total Locked TINY Amount (locked and not withdrawn)
- Total Power Count
    - Length of global power (total power) array.
- Last Total Power Timestamp
    - This timestamp is essential during the preparation of transactions. To streamline processes, it is denormalized and incorporated into the global state, effectively reducing the number of required data requests.

Boxes

1. {address}: AccountState
   Size: 32 bytes
   Cost: 28100

2. {address}{box_index}: [AccountPower]
   Size: 1008 (48 * 21) bytes
   Cost: 421700

3. tp{box_index}: [TotalPower]
   Size: 1008 (48 * 21) bytes
   Cost: 409700

4. sc{timestamp}: SlopeChange
   Size: 16 bytes
   Cost: 12900

Structs

Account State

Size: 32 bytes

```
AccountState{
        locked_amount (int)
        lock_end_time (int)
        power_count (int)
        deleted_power_count (int)
}
```

- Account state stores the current lock information and the length of the account power array.
- Calculating the current voting power of the account is possible using the Account State struct. Accessing the lastest Account Power struct is not required. A minor difference between Tiny Power calculated using Account State and Account Power because of the rounding errors.
- Locked amount and end time are set to 0 when the locked amount is withdrawn.
- Deleting an old stale account power is allowed, deleted power count increases when a box is deleted.

- The Account State stores current lock information and the length of the account power array.
- It enables the calculation of an account's current voting power directly using the Account State struct, eliminating the need to access the latest Account Power struct. However, due to rounding errors, a minor discrepancy might exist between the Tiny Power calculated using the Account State and that calculated using the Account Power.
- When the locked amount is withdrawn, both the locked amount and end time are set to 0.
- The system allows for the deletion of old, stale account power records. Consequently, the deleted power count increments whenever a box is deleted.

Tiny power calculation using account state:

$$slope = locked\ amount / max\ lock\ time\ (4\ years)$$
$$tiny\ power = slope * max(0, (lock\ end\ timestamp - target\ timestamp))$$

Account Power

Size: 48 bytes

AccountPower{
    bias (int)
    timestamp (int)
    slope (bytes[16])
    cumulative_power (bytes[16])
}

- Account Power stores the state changes following any action a governor takes concerning their locked assets and power.
- Account Powers are stored in an array format. This array structure is essential as it enables the calculation of both the voting power and cumulative power of a governor at any given time.

Tiny power calculation using account power:

$$if\ target\ timestamp > timestamp$$
$$time\ delta = target\ timestamp - timestamp$$
$$tiny\ power = bias - (time\ delta * slope)$$

Total Power

Size: 48 bytes

TotalPower{
    bias (int)
    timestamp (int)
    slope (bytes[16])
    cumulative_power (bytes[16])
}

- Total Power has the same structural design as Account Power.
- The primary function of Total Power is to facilitate the calculation of the total Tiny Power and cumulative power of all governors within the system.
- Total Power records are generated in actions taken by governors, such as creating or updating their locks. Additionally, Total Power records are also generated for each week's start and end timestamps, which are multiples of 604800.

Total tiny power calculation using account power if timestamps belong to the same week:

$$if\ target\ \ timestamp\ > timestamp$$
$$time\ delta\ = target\ timestamp\ -\ timestamp$$
$$total\ tiny\ power\ =\ bias\ -\ (time\ delta\ *\ \ slope)$$

Slope Change

Size: 16 bytes

SlopeChange{
    slope_delta (bytes[16])
}

- This represents the decrease in total voting power slope (Tiny Power) at grouped lock lock end timestamps.

If the target belongs to the next week, slope change must be applied to the total tiny power calculation:

$$time\ delta\ 1\ =\ end\ of\ week\ timestamp\ -\ \ timestamp$$
$$time\ delta\ 2\ =\ target\ timestamp\ -\ \ end\ of\ week\ timestamp$$
$$new\ slope\ =\ \ slope\ +\ \ slope\ delta\ (at\ the\ end\ of\ the\ week)$$
$$total\ tiny\ power\ =\ \ bias\ -\ (time\ delta\ 1\ *\ \ slope)\ -\ (time\ delta\ 2\ *\ \ new\ slope)$$

Methods

Init

This is a permissionless method and requires a one-time execution to initialize the Vault App.

Transaction Group:

1. Pay storage cost
   Sender: User Address
   Receiver: Vault App Address

2. App Call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“init”]
   Foreign Assets: [TINY ID]
   Fee: (2 * min fee)

3. App Call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“increase_budget”]

Logs:

- "init()"

Create Checkpoints

This function generates a checkpoint (total power) for the current time. If the most recent checkpoint is not from the current week, it will first create checkpoints for the previous weeks. However, due to budget constraints, a single method call can only generate a maximum of 9 total powers. If the last checkpoint is significantly outdated, multiple create checkpoint application calls may be necessary. Typically, this situation is not anticipated as other operations also generate checkpoints.

Transaction Group:

1. Pay storage cost (Maybe: If the total power array doesn’t have enough space)
   Sender: User Address
   Receiver: Vault App Address

2. App Call:

Sender: User Address
Index: Vault App ID
OnComplete: NoOp
App Args: [“create_checkpoints”]

3. App Call: increase_budget (Maybe: SDK calculates the number of checkpoint will be created and increase the opcode budget required times)
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“increase_budget”]

Logs:

- "total_power(uint64,uint64,uint64,uint128,uint128)" (index,bias,timestamp,slope,cumulative_power) (N times)
- "create_checkpoints()"

[Reference implementation from veCRV]

Create Lock

This function serves as the entry point for becoming a governor. Accounts that do not already have a lock can create one. To do this, TINY tokens are transferred to the application account, and the lock end timestamp is provided as a parameter in the application call. If an account already has a lock, they can modify their position using the "increase lock amount" and "extend lock end time" methods. In cases where an existing lock has expired, the account must first withdraw, and then proceed to create a new lock.

Transaction Group:

1. Pay storage cost (Maybe: If there isn’t available box space)
   Sender: User Address
   Receiver: Vault App Address

2. Axfer: Transfer TINY
   Sender: User Address
   Receiver: Vault App Address
   Index: TINY ID
   Amount: Lock Amount

3. App Call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp

App Args: [“create_lock”, <lock_end_timestamp>]
Foreign Assets: [TINY ID]

4. App Call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“increase_budget”]

Logs:

- "account_power(address,uint64,uint64,uint64,uint128,uint128)" (acccount_address,index,bias,timestamp,slope,cumulative_power)
- "total_power(uint64,uint64,uint64,uint128,uint128)" (index,bias,timestamp,slope,cumulative_power) (N times)
- "slope_change(uint64,uint128)" (timestamp,slope)
- "create_lock(address,uint64,uint64)" (user_address,locked_amount,lock_end_timestamp)

Increase Lock Amount

This method permits governors to add more TINY to their existing lock without altering the lock's end timestamp. It provides flexibility for governors to enhance their locked amount, thereby potentially increasing their Tiny Power, while maintaining their original lock duration.

Transaction Group:

1. Pay storage cost (Maybe: If there isn’t available box space)
   Sender: User Address
   Receiver: Vault App Address

2. Axfer: Transfer TINY
   Sender: User Address
   Receiver: Vault App Address
   Index: TINY ID
   Amount: Lock Amount

3. App Call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“increase_lock_amount”]

4. App Call:

Sender: User Address
Index: Vault App ID
OnComplete: NoOp
App Args: [“increase_budget”]

Logs:

- "account_power(address,uint64,uint64,uint64,uint128,uint128)" (acccount_address,index,bias,timestamp,slope,cumulative_power)
- "total_power(uint64,uint64,uint64,uint128,uint128)" (index,bias,timestamp,slope,cumulative_power) (N times)
- "slope_change(uint64,uint128)" (timestamp,slope)
- "increase_lock_amount(address,uint64,uint64,uint64)" (user_address,locked_amount,lock_end_timestamp,amount_delta)

Extend Lock End Time

This function enables governors to extend the duration of their lock without modifying the locked amount. It offers the flexibility to extend the lock period, which can potentially augment the governor's Tiny Power, while keeping the initial locked amount unchanged.

Transaction Group:

1. Pay storage cost (Maybe: If the total power array doesn’t have enough space)
   Sender: User Address
   Receiver: Vault App Address

2. App call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“extend_lock_end_time”, <lock_end_time>]

3. App Call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“increase_budget”]

Logs:

- "account_power(address,uint64,uint64,uint64,uint128,uint128)" (acccount_address,index,bias,timestamp,slope,cumulative_power)
- "total_power(uint64,uint64,uint64,uint128,uint128)" (index,bias,timestamp,slope,cumulative_power) (N times)
- "slope_change(uint64,uint128)" (timestamp,slope) (Revert old slope)
- "slope_change(uint64,uint128)" (timestamp,slope) (Update/Create)

- "extend_lock_end_time(address,uint64,uint64,uint64)"
  (user_address,locked_amount,lock_end_timestamp,time_delta)

Withdraw

Governors can withdraw their locked funds once the lock end timestamp has been reached. This method does not necessitate an update to the total power, unlike other methods, as the withdrawal does not impact the total power calculation.

Transaction Group:

1. App call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“withdraw”]
   Fee: (2 * min_fee)

Logs:

- "account_power(address,uint64,uint64,uint64,uint128,uint128)"
  (acccount_address,index,bias,timestamp,slope,cumulative_power)
- "withdraw(address,uint64)"
  (user_address, amount)

Delete Account Power Boxes

An account power box holds 21 account power structs. Users cover a minimum balance requirement as part of the storage cost. This method enables the application to delete obsolete account power boxes, thereby releasing the storage cost associated with these boxes. The method requires that at least one account power data point remains; if not, the transaction will fail.

Logs:

- "delete_account_power_boxes(address,uint64,uint64)"
  (user_address,box_index_start,box_count)

Delete Account State

Once the funds are withdrawn, users who do not wish to re-lock their TINY can utilize this method to delete their account state and account power boxes, thereby reclaiming the associated storage costs. A transaction group can encompass a maximum of 128 (16 transactions * 8) box references. If a user possesses more boxes than this limit, the “delete_account_power_boxes” method should be called first, followed by the “delete_account_state” method for any remaining boxes.

Logs:

- "delete_account_state(address,uint64,uint64)" (user_address,box_index_start,box_count)

Read Only Methods

Note:

- ***_of_** means it is specific to an account*
- ***_at_** means it is specific to a time in the past*
    - *get_tiny_power_**of***
    - *get_tiny_power_**of_at***
    - *get_total_tiny_power*
    - *get_total_tiny_power_**at***

Increase Budget

This method has been added to enable the increase of the opcode budget and the box read/write budget. It increases the **opcode budget 686*** and **box read budget 8KB**.

This method increases both the opcode budget and the box read/write budget. It increases the opcode budget by 686* and the box read budget by 8KB.

* The increase is not 700 because the transaction itself consumes 14 opcodes from the budget.

Get Box

Please see the common solutions.

Get TINY Power Of

This method provides the current TINY Power of a specified account.

Transaction Group:

1. App call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“get_tiny_power_of”, <address>]

Logs:

- return: uint64

Get TINY Power Of At

This method retrieves the TINY Power of an account at a specified past timestamp. It doesn't support future timestamps since the user may update their locks.

Transaction Group:

1. App call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“get_tiny_power_of_at”, <address>, <timestamp>, <account_power_index>]

Logs:

- return: uint64

Get Cumulative Power Of At

This method retrieves the Cumulative Tiny Power of an account at a specified past timestamp. It doesn't support future timestamps since the user may update their locks.

Transaction Group:

1. App call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“get_cumulative_power_of_at”, <address>, <timestamp>, <account_power_index>]

Logs:

- return: uint128

Get Total Tiny Power

This method provides the current total TINY Power.

Transaction Group:

1. App call: get_total_tiny_power
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“get_total_tiny_power”]

Logs:

- return: uint64

Get Total Tiny Power At

This method retrieves the total TINY Power at a specified past timestamp

Transaction Group:

1. App call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“get_total_tiny_power_at”, <timestamp>, <total_power_index>]

Logs:

- return: uint64

Get Cumulative Power At

This method retrieves the total Cumulative Tiny Power at a specified past timestamp

Transaction Group:

1. App call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“get_cumulative_power_at”, <timestamp>, <total_power_index>]

Logs:

- return: uint128

Get Account Cumulative Power Delta

This method returns the cumulative power difference between two specified timestamps. It is specifically designed for the Rewards App.

Transaction Group:

1. App call:
   Sender: User Address
   Index: Vault App ID
   OnComplete: NoOp
   App Args: [“get_account_cumulative_power_delta”, <address>, <timestamp 1>, <timestamp 2>, <account_power_index 1>, <account_power_index 2>]

Logs:

- return: uint128

Get Total Cumulative Power Delta

This method returns the total cumulative power difference between two specified timestamps. It is specifically designed for the Rewards App.

Transaction Group:

1. App call: get_total_cumulative_power_delta

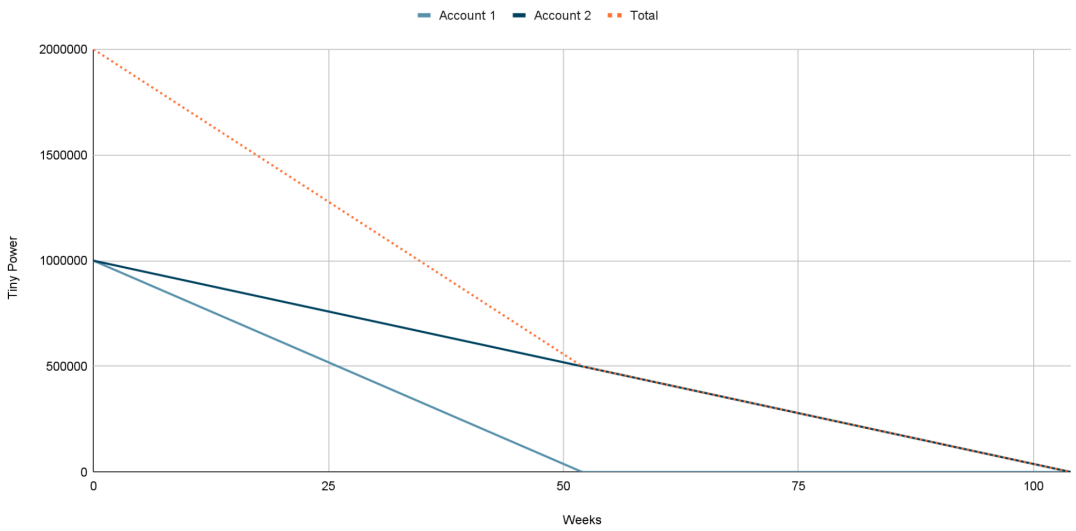
Sender: User Address
Index: Vault App ID
OnComplete: NoOp
App Args: [“get_account_cumulative_power_delta”, <timestamp 1>, <timestamp 2>, <total_power_index 1>, <total_power_index 2>]

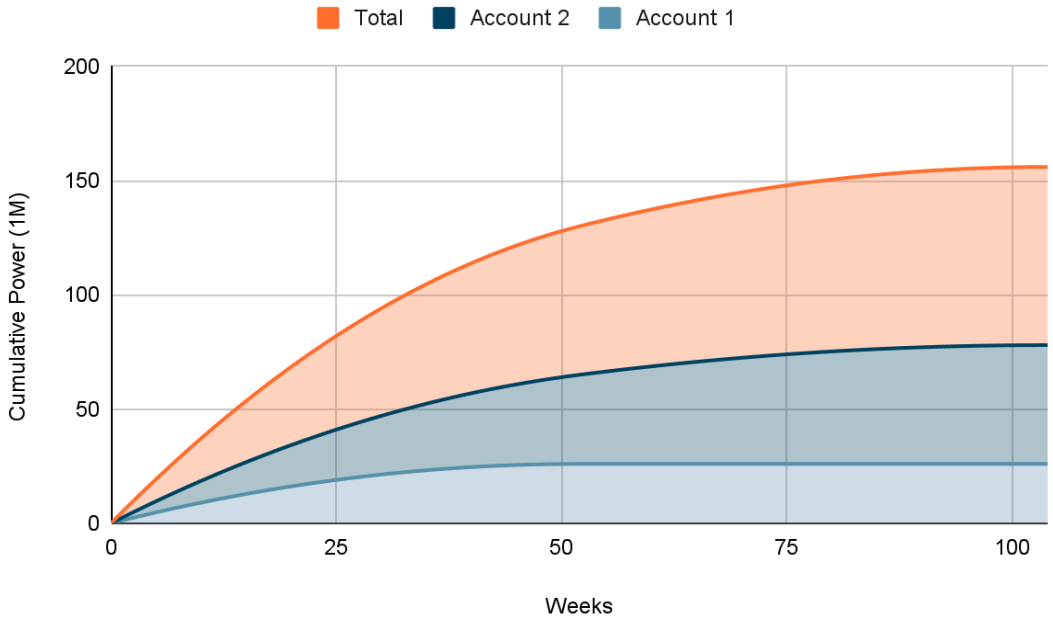
Logs:

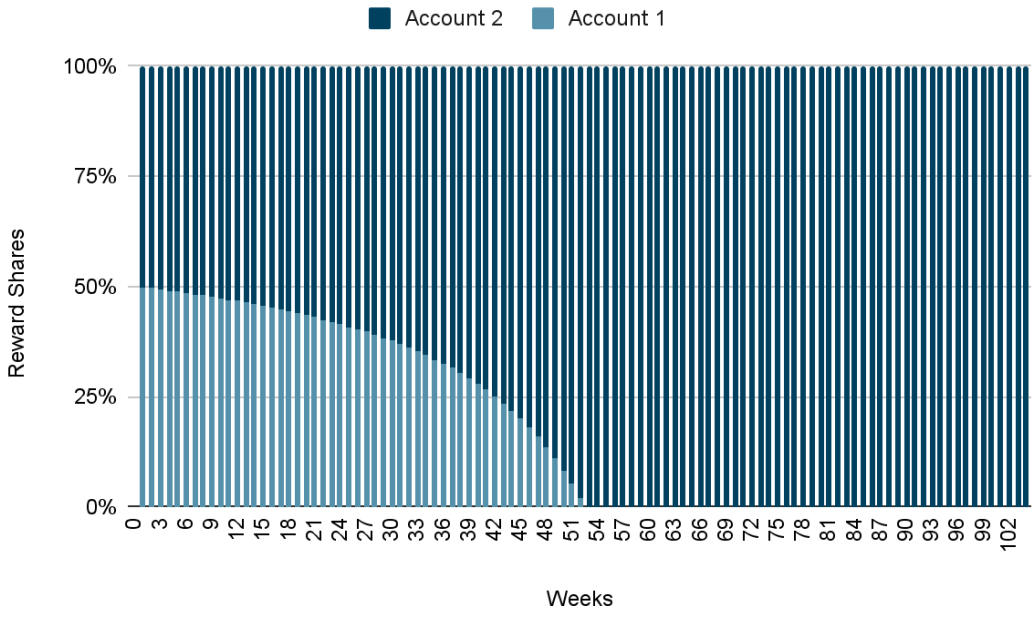
- return: uint128

Rewards App

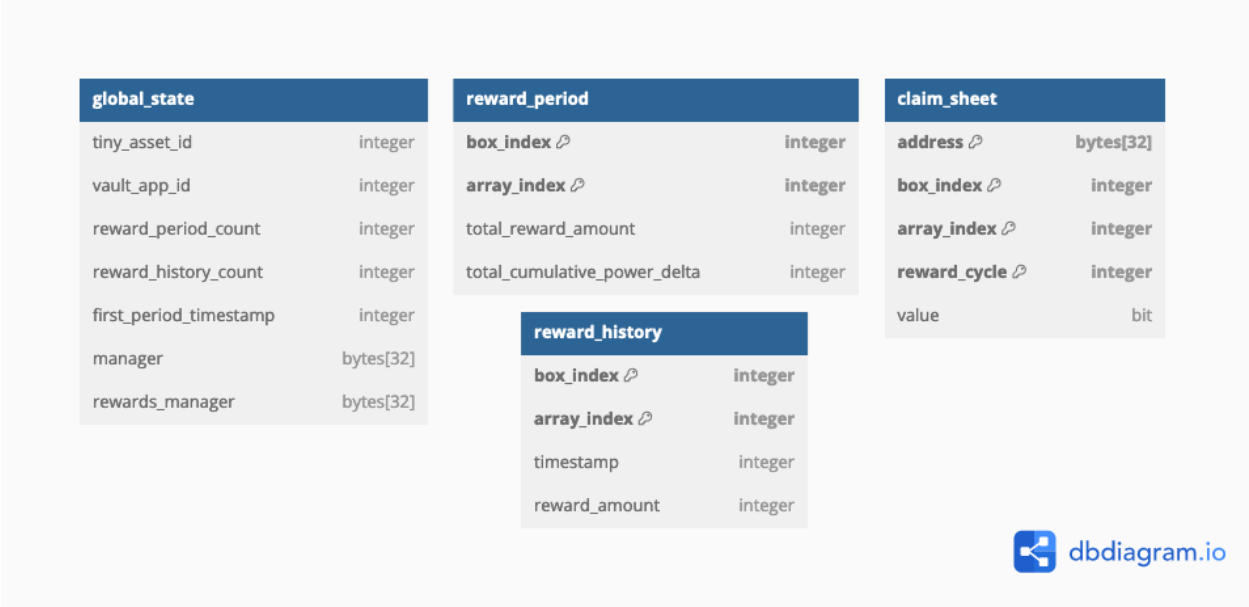
This application manages the distribution of Governance Rewards. Governors earn rewards proportional to their Tiny Power during weekly reward periods. The process involves calculating the cumulative tiny power across all accounts and determining each account's share. If a governor locks TINY for a longer duration, their cumulative power will be larger, resulting in an increased reward amount. The following diagrams present a simplified example of how rewards are allocated if there are only two governors: one with a lock of 4M TINY for 1 year and another with 2M TINY locked for 2 years. The account with the longer lock duration gains a larger share of the rewards compared to those with shorter lock periods.




Total
Account 2
Account 1
200
150
100
50
0
Cumulative Power (1M)
0
25
50
75
100
Weeks


Account 2
Account 1
100%
75%
50%
25%
0%
Reward Shares
0
3
6
9
12
15
18
21
24
27
30
33
36
39
42
45
48
51
54
57
60
63
66
69
72
75
78
81
84
87
90
93
96
99
102
Weeks

Storage Overview



https://dbdiagram.io/d/Rewards-App-649593e502bd1c4a5ef797f2

Global States

- Tiny Asset ID
- Vault App ID
- First Period Timestamp
    - Governors will begin accruing rewards based on their cumulative tiny power after this timestamp. This value is set to the start timestamp of the upcoming week by the init method.
- Reward History Count
    - It represents the length of the reward history array.
- Reward Period Count
    - It represents the length of the reward period array.
- Manager
    - This account holds the authority to update the contract, manage itself, and the rewards manager. By default, it is assigned to the creator of the contract. Ideally, this account should be a cold wallet for enhanced security.
- Rewards Manager
    - The Proposal Manager account holds the responsibility for initializing the application and determining the reward amounts.

Boxes

- rh{box_index}: [RewardHistory]
  Size: 256 (16*16) bytes
  Cost: 108900

- rp{box_index}: [RewardPeriod]
  Size: 1008 (24*42) bytes
  Cost: 409700

- c: {user_address}{box_index}
  Size: 1012 (8096 / 8) bytes
  Cost: 423400

  Left most Nth bit represents the claim status of the Nth index.

Structs

Reward History

Size: 16 bytes

RewardHistory{
    timestamp (int)

reward_amount (int)
}

- The rewards manager has the authority to modify the reward amount. All updates will be recorded historically.
- The current schedule is updating the reward amount annually. However, governors can propose changes to this schedule.
- This struct will contain the amount and the time of the change and it will be stored in an array.
- The updated reward amount will take effect for future periods.

Reward Period

Size: 24 bytes

RewardPeriod{
total_reward_amount (int)
total_cumulative_power_delta (bytes[16])
}

- A new reward period will be created at the end of each week. It will store the reward amount of the period and the total cumulative power delta.
- Total reward amount will be retrieved from the Reward History array. The last element before the period start timestamp will be used.
- Total cumulative power delta will be retrieved from the Vault Application using inner transactions.

Period Index Calculation

From start timestamp of week to period index:
$WEEK = 604800$
$Index = timestamp // WEEK - first\ period\ timestamp // WEEK$

From period index to start timestamp of week:
$period\ start\ timestamp = first\ period\ timestamp + index * WEEK$
$period\ end\ timestamp = period\ start\ timestamp + WEEK$

Methods

Init

This is a permissioned method and is required to be called once to initialize the Rewards App.

Transaction Group:

1. Pay storage cost

Sender: User Address
Receiver: Rewards App Address

2. App Call:
   Sender: User Address
   Index: Rewards App ID
   OnComplete: NoOp
   App Args: [“init”, <reward amount>]
   Foreign Assets: [TINY ID]
   Fee: (2 * min fee)

Logs:

- "reward_history(uint64,uint64,uint64)" (index,timestamp,reward_amount)
- "init(uint64,uint64)" (first_period_timestamp,reward_amount)

Create Reward Period

This is a permissionless method designed to create a reward period. It should be executed after a period (week) concludes. A prerequisite for this method is that a checkpoint must already be established for the end of the period on the Vault App. If such a checkpoint hasn't been created yet, the “create checkpoint” method of the Vault App must be called beforehand. It is assumed that this process will be managed by automated cron jobs.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Rewards App ID
   OnComplete: NoOp
   App Args: [“create_reward_period”, <total power period start index>, <total power period end index>, <reward history index>]
   Foreign Apps: [Vault App ID]
   Fee: (2 * min fee)

Logs:

- "reward_period(uint64,uint64,uint128)" (index,total_reward_amount,total_cumulative_power_delta)
- "create_reward_period(uint64,uint64,uint128)" (index,total_reward_amount,total_cumulative_power_delta)

Claim Rewards

This method stands as the most crucial method of the contract, allowing users to claim their rewards. The Rewards App retrieves the governor's cumulative power delta for a given period through an inner transaction from the Vault App. This method is optimized for bulk claims, particularly for claiming adjacent periods. A governor can claim rewards for up to 104 consecutive weeks (approximately 2 years).

Transaction Group:

1. Pay storage cost (Maybe, if a new claim sheet is required)
   Sender: User Address
   Receiver: Rewards App Address

2. App Call: increase_budget (Maybe N, if extra opcode budget or box read budget is required)

3. App Call:
   Sender: User Address
   Index: Rewards App ID
   OnComplete: NoOp
   App Args: [“claim_rewards”, <starting period index>, <period count>, <account_power_indexes (indexes are concatenated into a byte with fixed 8 bytes len)>]
   Foreign Apps: [Vault App ID]
   Fee: ((period_count + 2) * min fee)

Logs:

- "claim_rewards(address,uint64,uint64,uint64,uint64[])" (user_address,total_reward_amount,period_index_start,period_count,reward_amounts)

If the governor opts to claim rewards for N period, it necessitates N+1 inner transactions. This includes N transactions for retrieving data from the Vault App and 1 transaction for transferring the total reward amount to the governor's account.

For instance, if the governor seeks to claim rewards for periods 3, 4, 5, and 6, the parameters for the starting period index and the count of periods should be set to 3 and 4, respectively.

Set Reward Amount

This is a permissioned method, and only the rewards manager can invoke it. It establishes a new reward amount for upcoming periods.

1. Pay storage cost (Maybe, if the last reward history box is full)
   Sender: User Address
   Receiver: Rewards App Address

2. App Call:
   Sender: User Address
   Index: Rewards App ID
   OnComplete: NoOp
   App Args: [“set_reward_amount”, <reward amount>]

Logs:

- "reward_history(uint64,uint64,uint64)" (index,timestamp,reward_amount)
- "set_reward_amount(uint64,uint64)" (timestamp,reward_amount)

Get Box

Please see the common solutions.

Increase Budget

This method has been added to enable the increase of the opcode budget and the box read/write budget.

Transaction Group:

1. App call:
   Sender: User Address
   Index: Rewards App ID
   OnComplete: NoOp
   App Args: [“increase_budget”, <Optional: inner increase budget txn count>]
   Foreign Apps: [Vault App ID]

Note:

- *It allows increasing the opcode budget multiple times by calling the Increase Budget method of the Vault App using inner transactions, there is an additional argument for this purpose.*

Settings

The methods listed below are designed to update the setting variables.

Set Manager

This is a permissioned method, and only the manager can call it to modify the manager address itself.

1. App Call:

Sender: User Address
Index: Rewards App ID
OnComplete: NoOp
App Args: [“set_manager”, <address>]

Logs:

- "set_manager(address)"
  (new_manager)

Set Rewards Manager

This is a permissioned method, and only the manager can call it to modify the proposal manager address.

1. App Call:
   Sender: User Address
   Index: Rewards App ID
   OnComplete: NoOp
   App Args: [“set_rewards_manager”, <address>]

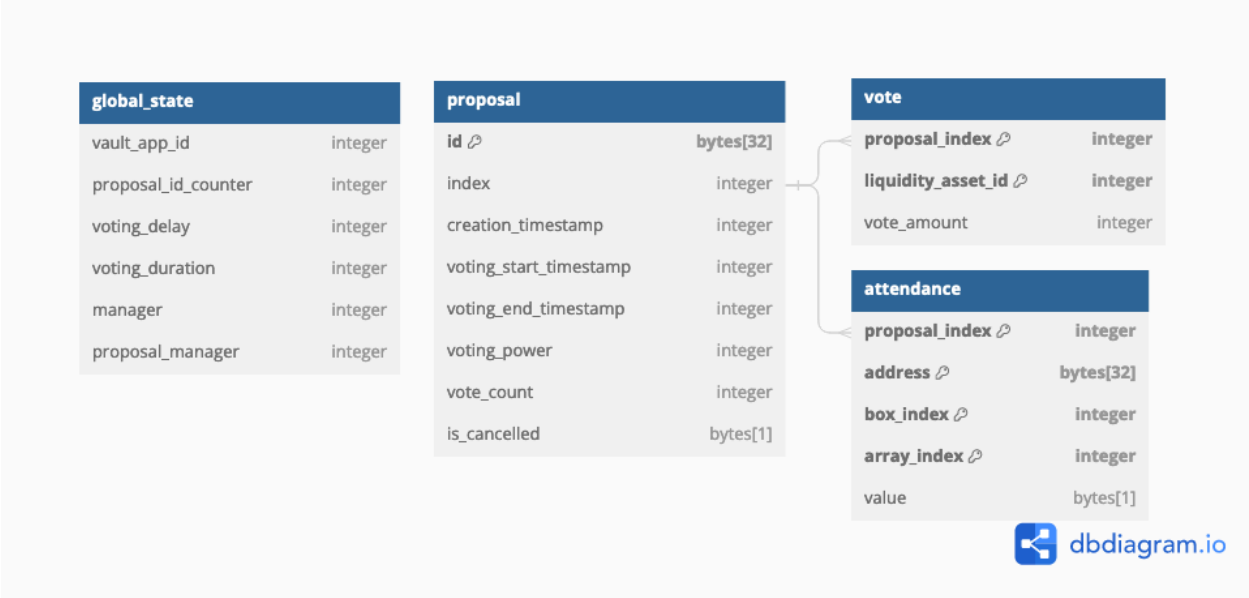
Logs:

- "set_rewards_manager(address)"
  (new_manager)

Staking Voting App

This application enables governors to select liquidity pools (V2) for distributing Liquidity Mining Rewards to liquidity providers. Each month, proposals for the upcoming Tinyman farming program will be created by the manager account. Governors can then allocate their voting power among the pools. Voting occurs on-chain. The outcome of each proposal will be analyzed off-chain, and based on these results, corresponding farming programs will be established.

Storage Overview



https://dbdiagram.io/d/Staking-Voting-App-64f1c91402bd1c4a5ecd9c74

*The number of pools is unlimited. For each proposal-pool pair, a separate vote box will be created.

Global States

- Vault App ID
- Proposal Index Counter
  This counter indicates the index of the next user proposal to be created. It begins at 0 and increments sequentially with each new proposal.
- Voting Delay
  - Refers to the interval between the starting of the voting process and the creation of the proposal. By default, this delay is set to 2 days. Voting starts at the first midnight after the addition of this delay period to the proposal creation time.
- Voting Duration
  - Refers to the time span, measured in days, between the start and end timestamps of the voting period. The default duration is set to 7 days.
- Manager
  - This account holds the authority to update the contract, manage itself, and the proposal manager. By default, it is assigned to the creator of the contract. Ideally, this account should be a cold wallet for enhanced security.
- Proposal Manager
  - The Proposal Manager account is responsible for managing daily administrative duties, which include creating and canceling tasks, as well as updating settings. This account routinely creates proposals according to a predefined schedule.

Boxes

1. p{proposal_id}: Proposal
   Size: 49 bytes
   Cost: 46100

2. v<proposal_index><liquidity_asset_id>: int, it stores the total voting power cast for proposal-asset pairs.
   Size: 8 bytes
   Cost: 12500

3. a<address><box_index>[<array_index>]: Each bit represents the attendance status of the account. The box index and array index parameters are derived based on the proposal index.
   Size: 24 bytes (192 bits)
   Cost: 28500
   Left most Nth bit represents the claim status of the Nth index.

Structs

Proposal

Struct size: 49 bytes

Proposal{
    index (int),
    creation_timestamp (int),
    voting_start_timestamp (int),
    voting_end_timestamp (int),
    voting_power (int),
    vote_count (int),
    is_cancelled (bytes[1])
}

Notes:

- *Voting Power: This represents the total voting power of all cast votes.*
- *Voting Count: This indicates the number of governors who have cast votes. Even if governors cast votes for multiple pools, the voting count increases by one for each governor.*

Methods

Create Proposal

This is a permissioned method, accessible exclusively to the proposal manager.

Transaction Group:

1. Pay storage cost
   Sender: User Address
   Receiver: Staking Voting App Address

2. App Call:
   Sender: User Address
   Index: Staking Voting App ID
   OnComplete: NoOp
   App Args: [“create_proposal”, <proposal id>]

Logs:

- "proposal(byte[59],uint64,uint64,uint64,uint64,uint64,uint64,bool)" (proposal_id,index,creation_timestamp,voting_start_timestmap,voting_end_timestmap,voting_power,voting_count,is_cancelled)
- "create_proposal(address,byte[59])" (proposer,proposal_id)

Cast Vote

Governors can only cast vote once and they cannot update the vote. They use the Tiny Power at the time the proposal is created. Updates on the lock amount or duration after the proposal creation time doesn't affect the valid Tiny Power on the proposal.

Governors can support multiple pools by sharing their Tiny Power among them. The transaction accepts liquidity asset ids and percentage values. It supports maximum of 16 asset ids and the total of percentage values must be 100. The total tiny powers will be saved to vote (proposal-asset) boxes.

Casting vote is allowed between voting start and end times. It is 7 days by default.

Transaction Group:

1. Pay storage cost (Maybe: If a user cast vote for a new proposal-pool pair or new attendance sheet is required)
   Sender: User Address
   Receiver: Staking Voting App Address

2. Appcall
   Sender: User Address
   Index: Staking Voting App ID
   App Args: ["cast_vote", <proposal_id>, <votes (byte array)>, <liquidity asset ids (byte array)>, <account_power_index>]
   Fee: (2 * min_fee)

3. Appcall (Maybe N: If extra box reference is needed)
   Sender: user_address
   Index: Staking Voting App ID
   App Args: ["increase_budget"]

Logs:

- "vote(uint64,uint64,uint64)" (asset_id,voting_power,vote_percentage)
- "proposal(byte[59],uint64,uint64,uint64,uint64,uint64,uint64,bool)" (proposal_id,index,creation_timestamp,voting_start_timestmap,voting_end_timestmap,voting_power,voting_count,is_cancelled)
- "cast_vote(address,byte[59],uint64)" (user_address,proposal_id,voting_power)

Notes:

- *votes (byte array) = <vote_1><vote_2>...<vote_N>*
- *liquidity asset ids (byte array) = <asset_id_1><asset_id_2>...<asset_id_N>*
- *If there are no liquidity pools for a given asset id, the transaction will still be processed successfully. However, the corresponding vote associated with that asset will be disregarded.*

Cancel Proposal

This is a permissioned method, accessible exclusively to the proposal manager. A proposal may be canceled at any time prior to the voting end timestamp.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Staking Voting App ID
   OnComplete: NoOp
   App Args: [“cancel_proposal”, <proposal id>]

Logs:

- "proposal(byte[59],uint64,uint64,uint64,uint64,uint64,uint64,bool)" (proposal_id,index,creation_timestamp,voting_start_timestmap,voting_end_timestmap,voting_power,voting_count,is_cancelled)
- "cancel_proposal(address,byte[59])" (manager_address,proposal_id)

Get Box

Please see the common solutions.

Increase Budget

This method has been added to enable the increase of the opcode budget and the box read/write budget.

Transaction Group:

1. App call:
   Sender: User Address
   Index: Staking Voting App ID
   OnComplete: NoOp
   App Args: [“increase_budget”, <Optional: inner increase budget txn count>]
   Foreign Apps: [Vault App ID]

Note:

- *It allows increasing the opcode budget multiple times by calling the Increase Budget method of the Vault App using inner transactions, there is an additional argument for this purpose.*

Settings

The methods listed below are designed to update the setting variables.

Set Voting Delay

This is a permissioned method, and only the proposal manager can call it to modify the voting delay.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Staking Voting App ID
   OnComplete: NoOp
   App Args: [“set_voting_delay”, <new_voting_delay>]

Logs:

- "set_voting_delay(uint64)" (new_voting_delay)

Set Voting Duration

This is a permissioned method, and only the proposal manager can call it to modify the voting duration.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Staking Voting App ID
   OnComplete: NoOp
   App Args: [“set_voting_duration”, <new_voting_duration>]

Logs:

- "set_voting_duration(uint64)" (new_voting_duration)

Set Manager

This is a permissioned method, and only the manager can call it to modify the manager address itself.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Staking Voting App ID
   OnComplete: NoOp

App Args: [“set_manager”, <address>]

Logs:

- "set_manager(address)" (new_manager)

Set Proposal Manager

This is a permissioned method, and only the manager can call it to modify the proposal manager address.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Staking Voting App ID
   OnComplete: NoOp
   App Args: [“set_proposal_manager”, <address>]

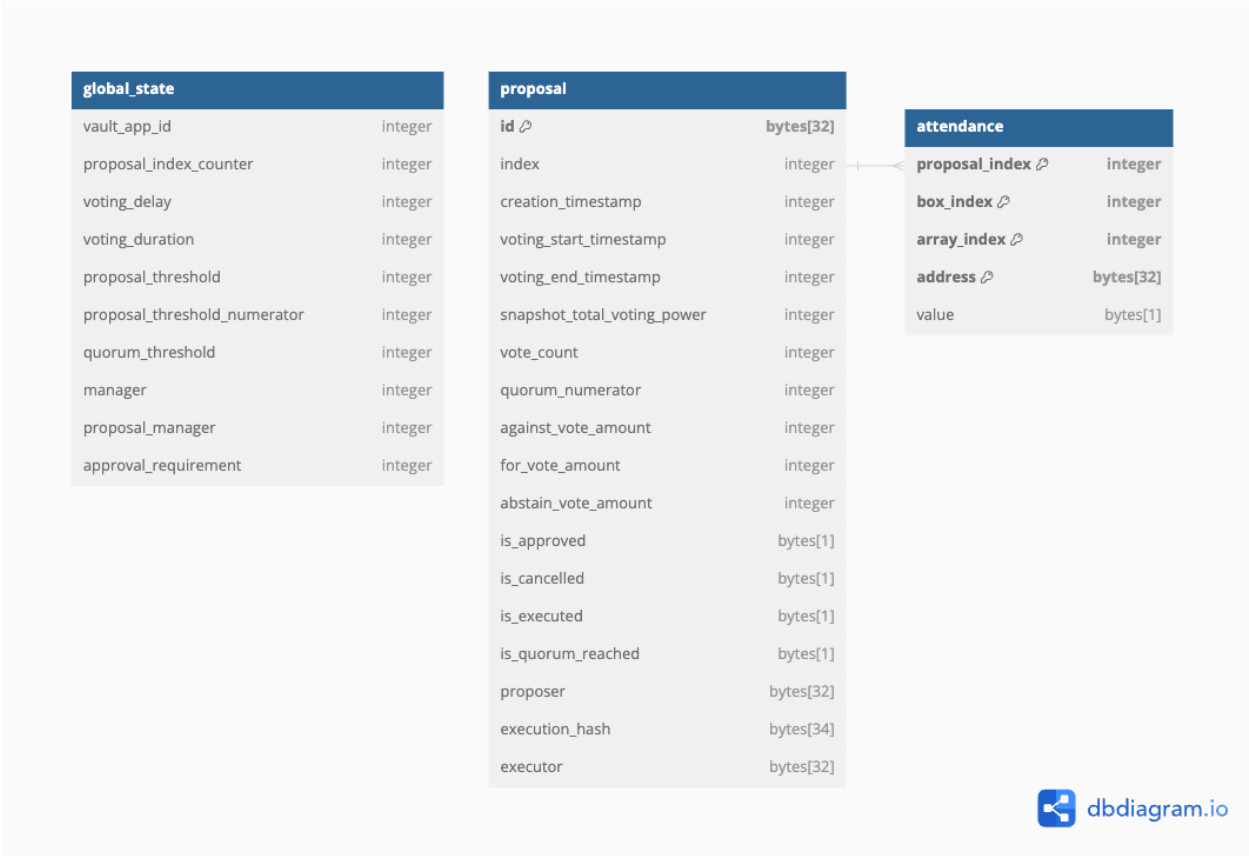
Logs:

- "set_proposal_manager(address)" (new_manager)

Proposal Voting App

The Proposal Voting App is designed to streamline the process of community-led decision-making pertaining to the governance and operational strategies of Tinyman. Eligible governors are empowered to submit their proposals, articulating the requisite resource allocation from the treasury or protocol management actions. During the designated voting period, all governors have the opportunity to indicate their support and engage with the proposals presented.

Storage Overview



https://dbdiagram.io/d/Proposal-Voting-App-64958ec402bd1c4a5ef75239

Global States

- Vault App ID
- Proposal Index Counter

    - This counter indicates the index of the next user proposal to be created. It begins at 0 and increments sequentially with each new proposal.
- Voting Delay
    - Refers to the interval between the starting of the voting process and the creation of the proposal. By default, this delay is set to 2 days. Voting starts at the first midnight after the addition of this delay period to the proposal creation time.
- Voting Duration
    - Refers to the time span, measured in days, between the start and end timestamps of the voting period. The default duration is set to 7 days.
- Proposal Threshold
    - This parameter specifies the minimum Tiny Power required for a governor to be eligible to create a proposal. The default threshold is set at 450K.
- Proposal Threshold Numerator
    - This parameter determines the minimum Tiny Power a governor must have to be eligible for proposal creation, expressed as a percentage of the total available Tiny Power. The default setting for this numerator is 0%, effectively disabling this requirement.
- Quorum Threshold
    - This represents the minimum Tiny Power necessary to achieve a quorum. The combined total of FOR and ABSTAIN votes must meet or exceed this threshold. Default value is 2,25M.
- Manager
    - This account holds the authority to update the contract, manage itself, and the proposal manager. By default, it is assigned to the creator of the contract. Ideally, this account should be a multisig backed by cold wallets for enhanced security.
- Proposal Manager
    - The Proposal Manager account is responsible for managing daily administrative duties, which include approving and canceling tasks, as well as updating settings.

Boxes

1. p<proposal_id>: Proposal
   Size: 116 bytes
   Cost: 99300

2. a<address><box_index>[<array_index>]: Each bit represents the attendance status of the account. The box index and array index parameters are derived based on the proposal index.
   Size: 24 bytes (192 bits)
   Cost: 28500
   Left most Nth bit represents the claim status of the Nth index.

Structs

Proposal

Struct size: 182 bytes

Proposal{
    index (int),
    creation_timestamp (int),
    voting_start_timestamp (int),
    voting_end_timestamp (int),
    snapshot_total_voting_power (int),
    vote_count (int),
    quorum_threshold (int),
    against_vote_amount (int),
    for_vote_amount (int),
    abstain_vote_amount (int),
    is_approved (bytes[1]),
    is_cancelled (bytes[1]),
    is_excuted (bytes[1]),
    is_quorum_reached (bytes[1]),
    proposer (bytes[32]),
    execution_hash (bytes[34])
    executor (bytes[32])
}

Methods

Create Proposal

Governors have the capability to create on-chain proposals using this method. In order to prevent spam, only governors possessing Tiny Power exceeding certain limits are permitted to create proposals. These limits are determined by: "proposal_threshold" and "proposal_threshold_numerator" variables stored in the global state.

- Proposal Threshold: This is a static variable representing a specific Tiny Power amount, which is stored in the global state. Governors must have Tiny Power equal to or greater than this predefined amount to create a proposal.

- Proposal Threshold Numerator: This dynamic limit is calculated as a proportion of the total Tiny Power available at the time of the proposal's creation. To determine this limit, the total power retrieved from the Vault App. The formula for calculating this proportional limit is: $limit = proposal\ threshold\ numerator * total\ tiny\ power / 100$This formula provides a balanced and adaptable threshold for proposal creation.

By employing these two limits, the system ensures that only eligible governors with sufficient Tiny Power can initiate proposals.

If the approval requirement setting is on, the proposal must be approved by the manager first. When the proposal is approved, the voting period (start and end timestamps) will be set to the proposal. If approval is not required, voting period will be set immediately within the creation transaction.

If the approval requirement setting is activated, the proposal must first receive approval from the manager. Once the proposal is approved, the voting period (the start and end timestamps) will be assigned to the proposal. In cases where approval is not required, the voting period will be set immediately within the creation transaction.

Transaction Group:

1. Pay storage cost
   Sender: User Address
   Receiver: Proposal Voting App Address

2. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“create_proposal”, <proposal id>, <execution_hash>, <executor>]

Logs:

- "proposal(byte[59],uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,bool,bool,bool,bool,address)"
  (proposal_id,index,creation_timestamp,voting_start_timestamp,voting_end_timestamp,snapshot_total_voting_power,vote_count,quorum_numerator,against_voting_power,for_voting_power,abstain_voting_power,is_approved,is_cancelled,is_executed,is_quorum_reached,proposer_address)
- "create_proposal(address,byte[59])
  (proposer_address,proposal_id)

Cancel Proposal

This is a permissioned method, accessible exclusively by the proposal manager. A proposal may be canceled at any time before the voting end timestamp.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp

App Args: [“cancel_proposal”, <proposal id>]

Logs:

- "proposal(byte[59],uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,bool,bool,bool,bool,address)"
  (proposal_id,index,creation_timestamp,voting_start_timestamp,voting_end_timestamp,snapshot_total_voting_power,vote_count,quorum_numerator,against_voting_power,for_voting_power,abstain_voting_power,is_approved,is_cancelled,is_executed,is_quorum_reached,proposer_address)
- "cancel_proposal(address,byte[59])
  (manager_address,proposal_id)

Approve Proposal

This is a permissioned method, and only the proposal manager has the authority to call it. The manager approves the proposal if the proposal is valid; otherwise, the proposal is canceled. If approval is not required for the proposal, there is no need for an approval call.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“approve_proposal”, <proposal id>]

Logs:

- "proposal(byte[59],uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,bool,bool,bool,bool,address)"
  (proposal_id,index,creation_timestamp,voting_start_timestamp,voting_end_timestamp,snapshot_total_voting_power,vote_count,quorum_numerator,against_voting_power,for_voting_power,abstain_voting_power,is_approved,is_cancelled,is_executed,is_quorum_reached,proposer_address)
- "approve_proposal(address,byte[59])
  (manager_address,proposal_id)

Cast Vote

All governors having Tiny Power at the snapshot time (the moment of proposal creation) are eligible to cast a vote. They must choose only one option, and once cast, votes cannot be updated.

Vote Enum:

- *0: Against*
- *1: For*
- *2: Abstain*

Transaction Group:

1. Pay storage cost (Optional: This applies if there is no attendance sheet that includes the proposal index.)
   Sender: User Address
   Receiver: Proposal Voting App Address

2. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“cast_proposal”, <proposal id> <vote [0, 1, 2]>, <account_power_index>]

Logs:

- "proposal(byte[59],uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,bool,bool,bool,bool,address)" (proposal_id,index,creation_timestamp,voting_start_timestamp,voting_end_timestamp,snapshot_total_voting_power,vote_count,quorum_numerator,against_voting_power,for_voting_power,abstain_voting_power,is_approved,is_cancelled,is_executed,is_quorum_reached,proposer_address)
- "cast_vote(address,byte[59],uint64,uint64)" (governor_address,proposal_id,vote,voting_power)

Execute Proposal

This method should be called by the proposal executor address after the execution of the proposal to update the on-chain state of the proposal.

Transaction Group:

1. App Call:
   Sender: Executor Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“execute_proposal”, <proposal id>]

Logs:

- "proposal(byte[59],uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,bool,bool,bool,bool,address)" (proposal_id,index,creation_timestamp,voting_start_timestamp,voting_end_timestamp,snapshot_total_voting_power,vote_count,quorum_numerator,against_voting_power,for_voting_power,abstain_voting_power,is_approved,is_cancelled,is_executed,is_quorum_reached,proposer_address)
- "execute_proposal(address,byte[59])" (manager_address,proposal_id)

Read Only Methods

Get Proposal

This function enables other on-chain contracts to read the state of a proposal.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“get_proposal”, <proposal id>]

Logs:

- return:
  byte[59],uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,uint64,bool,bool,bool,bool,address
  (proposal_id,index,creation_timestamp,voting_start_timestamp,voting_end_timestamp,snapshot_total_voting_power,vote_count,quorum_numerator,against_voting_power,for_voting_power,abstain_voting_power,is_approved,is_cancelled,is_executed,is_quorum_reached,proposer_address), uint64

Proposal State Enum:

- 0: Waiting For Approval. The proposal has been created and is awaiting approval.
- 1: Cancelled. The proposal has been cancelled by the manager.
- 2: Pending. The voting period has not yet started.
- 3: Active. The proposal is within the voting period; it has started but not yet ended.
- 4: Defeated. The voting period is over, and the 'for' votes are less than or equal to 'against' votes, or the quorum has not been reached.
- 5: Succeeded. The voting period is over and the 'for' votes are greater than 'against' votes, and the quorum has been reached.
- 6: Executed. The required actions as a result of the proposal have been executed.

Has voted

This function allows other on-chain contracts to verify whether a governor has cast a vote or not.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“has_voted”, <proposal id>, <address>]

Logs:

- return: uint64

Get Box

Please see the common solutions.

Increase Budget

This method has been added to enable the increase of the opcode budget and the box read/write budget.

Transaction Group:

1. App call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“increase_budget”, <Optional: inner increase budget txn count>]
   Foreign Apps: [Vault App ID]

Note:

- *It allows increasing the opcode budget multiple times by calling the Increase Budget method of the Vault App using inner transactions, there is an additional argument for this purpose.*

Settings

The methods listed below are designed to update the setting variables.

Set Voting Delay

This is a permissioned method, and only the proposal manager can call it to modify the voting delay.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“set_voting_delay”, <new_voting_delay>]

Logs:

- "set_voting_delay(uint64)" (new_voting_delay)

Set Voting Duration

This is a permissioned method, and only the proposal manager can call it to modify the voting duration.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“set_voting_duration”, <new_voting_duration>]

Logs:

- "set_voting_duration(uint64)" (new_voting_duration)

Set Proposal Threshold

This is a permissioned method, and only the proposal manager can call it to modify the proposal threshold.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“set_proposal_threshold”, <new_proposal_threshold>]

Logs:

- "set_proposal_threshold(uint64)" (new_voting_delay)

Set Proposal Threshold Numerator

This is a permissioned method, and only the proposal manager can call it to modify the proposal threshold numerator.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“set_proposal_threshold_numerator”, <new_proposal_threshold_numerator>]

Logs:

- "set_proposal_threshold_numerator(uint64)"

Set Quorum Threshold

This is a permissioned method, and only the proposal manager can call it to modify the quorum threshold.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“set_quorum_threshold”, <new_quorum_numerator>]

Logs:

- "set_quorum_threshold(uint64)"

Disable Approval Requirement

This is a permissioned method, and only the proposal manager has the authority to call it for the purpose of disabling the approval requirement. Once disabled, it is important to note that the proposal manager cannot re-enable it. However this functionality could technically be re-enabled through an application update by the manager.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“disable_approval_requirement”]

Logs:

- "disable_approval_requirement()"

Set Manager

This is a permissioned method, and only the manager can call it to modify the manager address itself.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“set_manager”, <address>]

Logs:

- "set_manager(address)" (new_manager)

Set Proposal Manager

This is a permissioned method, and only the manager can call it to modify the proposal manager address.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Proposal Voting App ID
   OnComplete: NoOp
   App Args: [“set_proposal_manager”, <address>]

Logs:

- "set_proposal_manager(address)" (new_manager)

Executors (Contracts)

NOTE: This section describes functionality that has not been implemented yet. It is included here to give an idea of how the governance system can operate in a fully on-chain and permissionless manner in the future.

Executors are contracts that can execute the actions of successful governance proposals. Tinyman Governance comprises three executors which are Algorand applications (smart contracts): Arbitrary, Fee Management, Treasury Management.

Any account that is rekeyed to one of these executor contracts will be fully controlled by the governance process. It is expected that these will eventually include all accounts currently controlled by the Tinyman Multisig.

Arbitrary Executor makes execution of arbitrary transactions possible, giving governors a flexible way to make changes about Governance and Protocol. It lets governance make transactions on behalf of well known Tinyman accounts.

Fee Management Executor gives governance the ability to make changes in Protocol, AMM fees, who can collect, set and manage fees.

Treasury Management Executor controls the distribution of protocol revenue and TINY token. Governors can fund more complex processes through this contract and make implementing features possible.

Execution heavily relies on proposal data kept on Proposal Voting App's boxes. Especially on proposal's `execution_hash` and `executor` fields. `execution_hash` is used for validation purposes by executor apps. `execution_hash` should verify user's transaction(s).

Arbitrary Executor App

Governors can execute their transactions and transaction group which are pre-determined and their hashes attached to proposal.

Usage Flow

It is intended that this executor would be used with protocol related accounts that need to allow for flexible proposals involving arbitrary transactions. This may include the deployment of apps, creation of ASAs, among other use cases.
It is planned that this method would be used with Arbitrary Executor LogicSig, which is a way to delegate permission of Protocol's known accounts such as the one with "tinyman.algo" NFDomain name. If the governor wants to send transactions on behalf of these accounts, governor needs to prepare a future valid transaction from the relevant account and add it to the transaction group after an App Call that invokes one of the validate methods. The transaction id or transaction group id should be included in the `execution_hash` of the proposal. When/if the proposal passes and when the valid round window arrives the transaction group can be submitted on chain with the aforementioned logic signature.

Methods

Validate Transaction

Governors have the capability to execute arbitrary transactions using this method.

Requirements:
Proposal's `execution_hash` field should be in following format:

"vt" + base32decode(<arbitrary_transaction_id>)

Transaction Group:

1. App Call:

Sender: User Address
Index: Arbitrary Executor App ID
OnComplete: NoOp
App Args: [“validate_transaction”, proposal_id]
Foreign Apps: [Proposal Voting App ID]

2. Arbitrary Transaction: Can be any type of transaction.

Validate Group

Governors have the capability to execute arbitrary transaction groups using this method.

Requirements:
Proposal’s `execution_hash` field should be in following format:

“vg” + base32decode(<arbitrary_transaction_group_id>)
This should be the group id of the full group including the application call.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Arbitrary Executor App ID
   OnComplete: NoOp
   App Args: [“validate_group”, proposal_id]
   Foreign Apps: [Proposal Voting App ID]

2 - 15. Arbitrary Transactions: Can be any type of transactions.

Fee Management Executor App

Governors can change fees for a specific pool or change special addresses: Fee Setter, Fee Manager and Fee Collector.

Usage Flow

It is intended that this contract would be used for governing AMM Protocol V2. Most of the functionalities come from already implemented AMM methods. Special addresses could be changed by The Fee Manager but with Fee Management Executor governors can get these permissions. For example, governor wants some account A to be the new Fee Setter. Governor should create a “set_fee_setter” transaction as specified below, prepare an `execution_hash` accordingly and create a proposal. After the proposal’s success, governor can send this transaction and execute the proposal.

Methods

Set Fee Setter

Governors have the capability to change the “fee setter” on AMM contract.

Requirements:
Proposal’s `execution_hash` field should be in following format:
“fs” + sha256(“set_fee_setter” + decode_address(<new_fee_setter>))

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Fee Management Executor App ID
   OnComplete: NoOp
   App Args: [“set_fee_setter”, <proposal_id>, <new_fee_setter>]
   Foreign Apps: [Proposal Voting App ID, AMM V2 App ID]

Set Fee Manager

Governors have the capability to change the “fee manager” on AMM contract.

Requirements:
Proposal’s `execution_hash` field should be in following format:
“fm” + sha256(“set_fee_manager” + decode_address(<new_fee_manager>))

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Fee Management Executor App ID
   OnComplete: NoOp
   App Args: [“set_fee_manager”, <proposal_id>, <new_fee_manager>]
   Foreign Apps: [Proposal Voting App ID, AMM V2 App ID]

Set Fee Collector

Governors have the capability to change the “fee collector” on AMM contract.

Requirements:
Proposal’s `execution_hash` field should be in following format:
“fc” + sha256(“set_fee_collector” + decode_address(<new_fee_collector>))

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Fee Management Executor App ID
   OnComplete: NoOp
   App Args: [“set_fee_collector”, <proposal_id>, <new_fee_collector>]
   Foreign Apps: [Proposal Voting App ID, AMM V2 App ID]

Set Fee For Pool

Governors have the capability to change the “total fee share” and “protocol fee rate” parameters for a pool in AMM.

Requirements:
Proposal’s `execution_hash` field should be in following format:
“sf” + sha256(“set_fee_for_pool” + decode_address(<pool_address>) + itob(<pool_total_fee_share>) + itob(<pool_protocol_fee_ratio>))

Transaction Group:
1. App Call:
   Sender: User Address
   Index: Fee Management Executor App ID
   OnComplete: NoOp
   App Args: [“set_fee_for_pool”, <proposal_id>, <pool_address>, <pool_total_fee_share>, <pool_protocol_fee_rate>]
   Foreign Apps: [Proposal Voting App ID, AMM V2 App ID]

Treasury Management Executor App

Governors can send Algo or any ASA from the protocol treasury accounts to any accounts.

Usage Flow

It is intended for this contract to be able to fund future development. For example; if governors reach a quorum to make a new feature or change that needs workforce, with this contract it is possible to fund development.

Methods

Send

Governors have the capability to execute arbitrary transfers using this method.

Requirements:
Proposal's `execution_hash` field should be in following format:
"sn" + sha256("send" + decode_address(<treasury_account>) + decode_address(<receiver>) + itob(<amount>) + itob(<asset_id>))

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Treasury Management Executor App ID
   OnComplete: NoOp
   App Args: ["send", <proposal_id>, <treasury_account>, <receiver>, <amount>, <asset_id>]
   Foreign Apps: [Proposal Voting App ID]
   Accounts: [<treasury_account>, <treasure_receiver>]