Tokens

Introduction

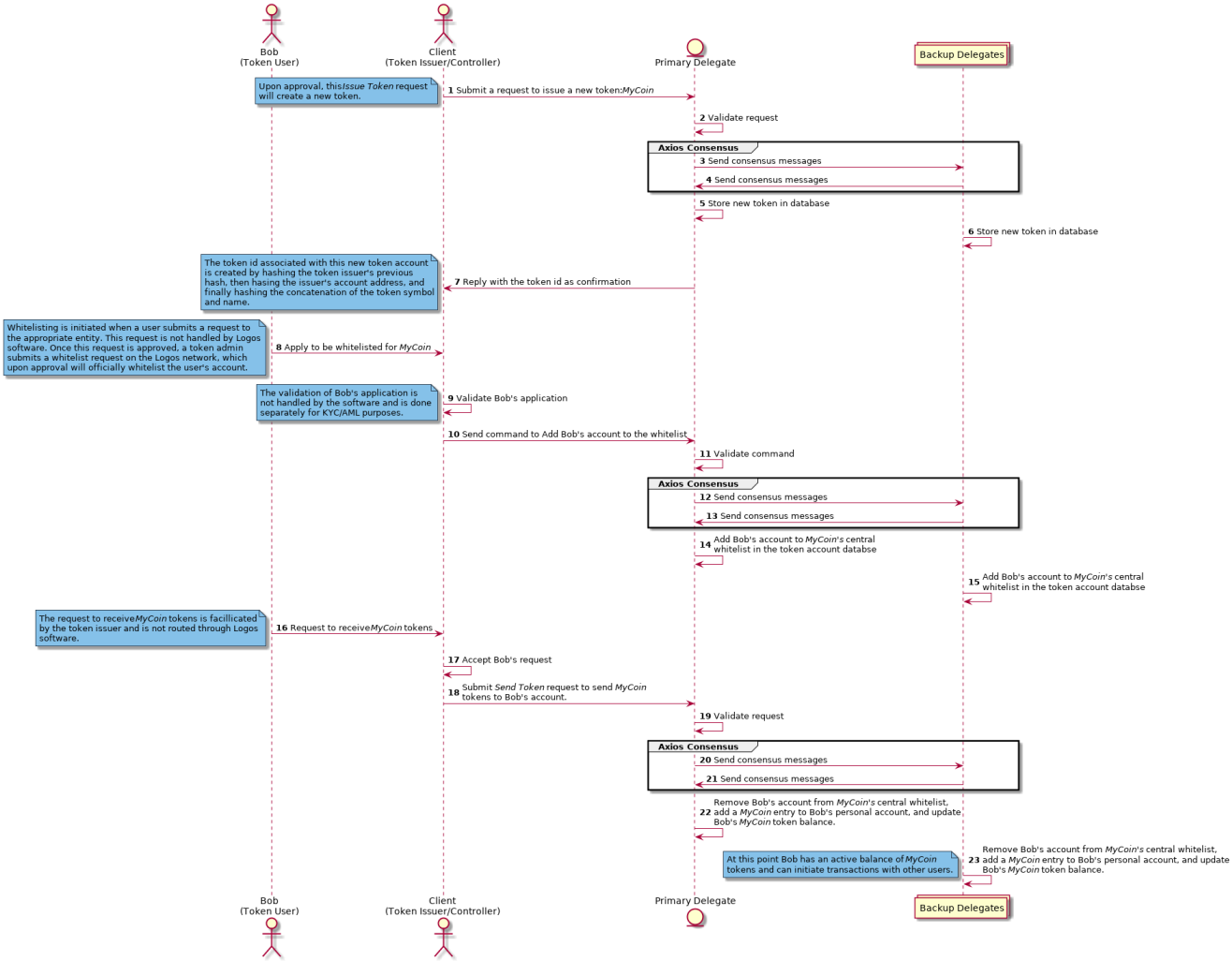
The Logos token platform is an extension of the Logos Network in that it supports the issuance, management and daily operations of tokens on top of the Logos Network architecture. All token operations are initiated by requests that are submitted to delegates and then approved via Axios Consensus. As the token platform is an extension of the existing Logos Network architecture, this section assumes that the reader has a basic understanding of that underlying system, and some details may be omitted.

Token accounts are extensions of conventional Logos accounts in that they have a balance that can be increased or decreased by making transactions on the network, resulting in a historical chain of operations. Additionally, token accounts contain settings that dictate which operations the token does or does not support. These settings also have mutability options that dictate whether or not the settings can be changed. For example, if the total supply for a particular token is set as immutable, then additional tokens cannot be issued. Furthermore, operations performed on a token account are chained together (as with transactions between regular Logos user accounts) with each operation referring to the hash of the request that preceeded it. The following sections describe the Logos token platform in greater detail.

Token Issuance

Token issuance is the process by which a new token is created. To create a new token, an issuer submits a Token Issuance Request, which contains information that dictates the operation of the new token on the Logos Network. Once the request is confirmed, a the token will be available for participants per rules dictated by the token's settings.

Issue and Request Tokens



Freezelist

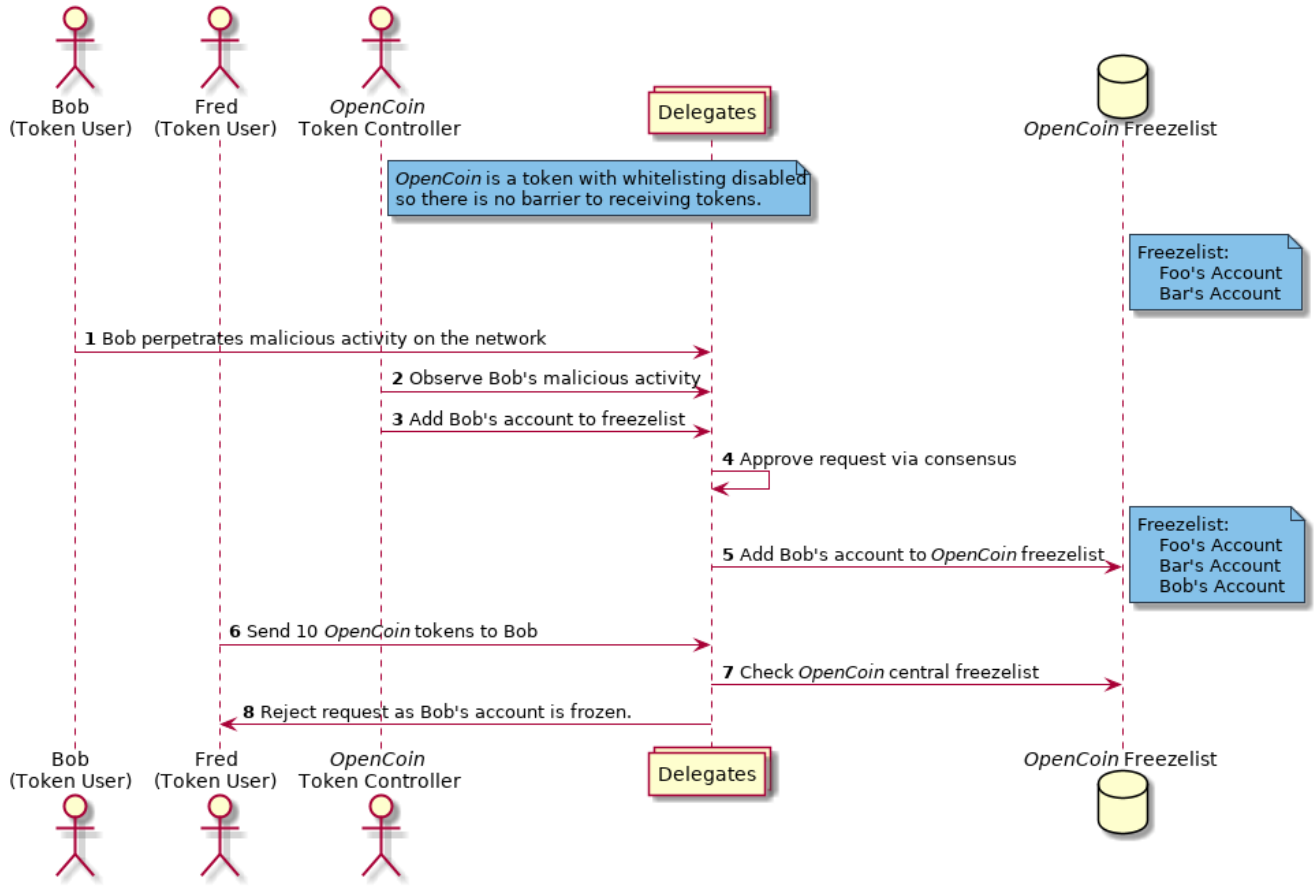
Freezing an account prevents it from both sending and receiving tokens and is enforced per token type: If an account that has tokens for MyCoin and YourCoin gets frozen by a MyCoin controller, the account can still use its YourCoin tokens. Furthermore, requests to send tokens to or from frozen accounts will be rejected. Regarding the implementation, the freezelist was originally to be represented as a central list containing the address of each frozen account, but this approach has the following drawbacks:

- This can result in database bloat if many accounts are frozen.
- Nodes may have to scan through many entries to confirm whether a particular account is or isn't frozen, impacting performance.

One solution is to simply use a single bit in the token entry stored within an individual account to indicate whether this account is frozen. However, a token account controller may want to freeze accounts that have not yet been tethered to the token (an account is tethered by being whitelisted by a controller or by receiving tokens). To allow this we would have to enable a controller to add a token entry to an untethered account in order to set the appropriate bit, however, this can be exploited by a malicious controller who

spams the network by freezing every account, permanently increasing the size of each account. To mitigate this, a central freezelist will be used for untethered accounts. Since the users' accounts themselves will be used to implement the freezelist, the central freezelist doesn't need to be checked for transactions between accounts with preexisting token entries. The only time the central freezelist needs to be consulted is when an untethered account is about to receive tokens for the first time. When this occurs, the freezelist will be checked, and if the account in question is listed therein, the request will be rejected.

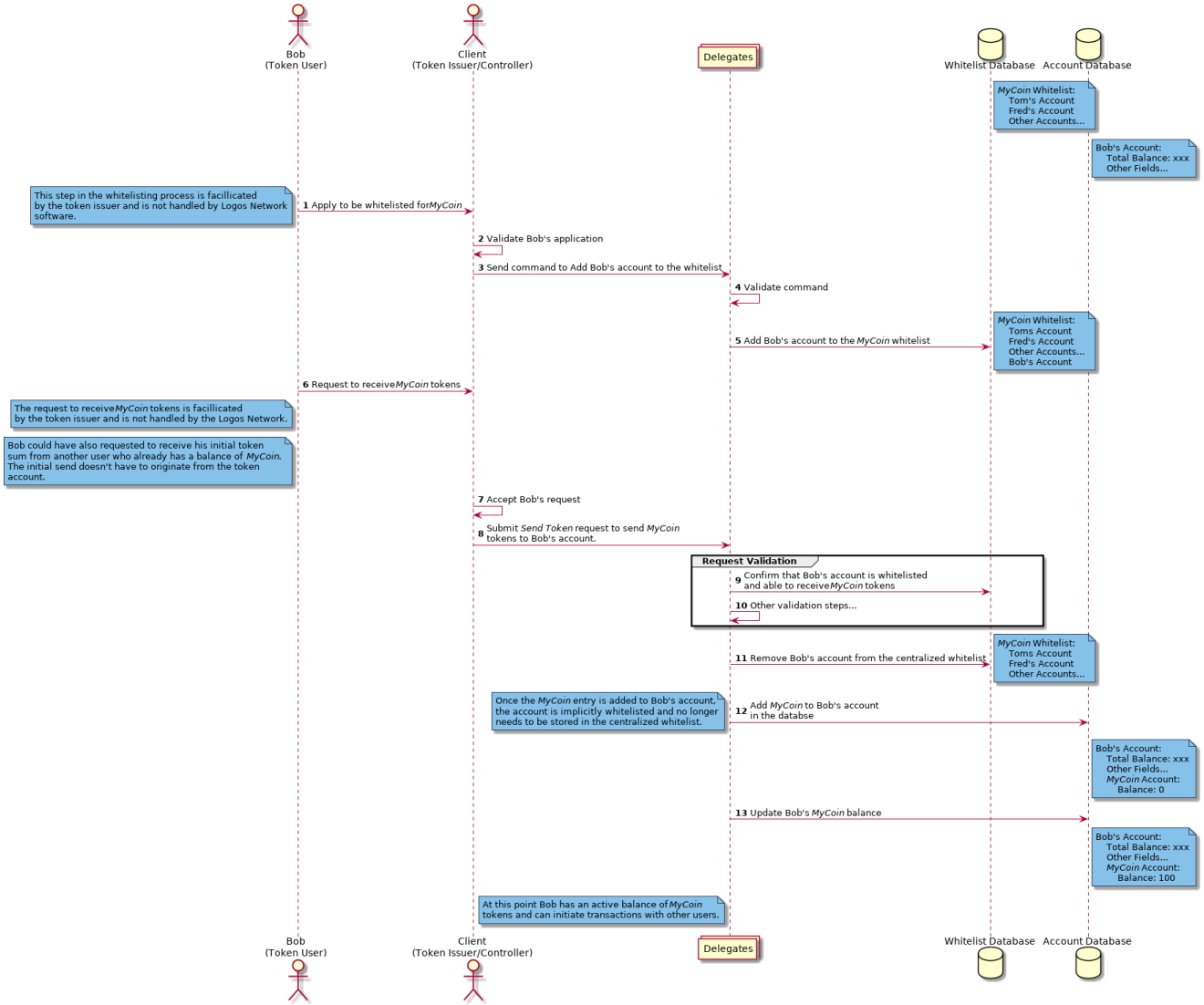
Freezing an Untethered Account



Whitelist

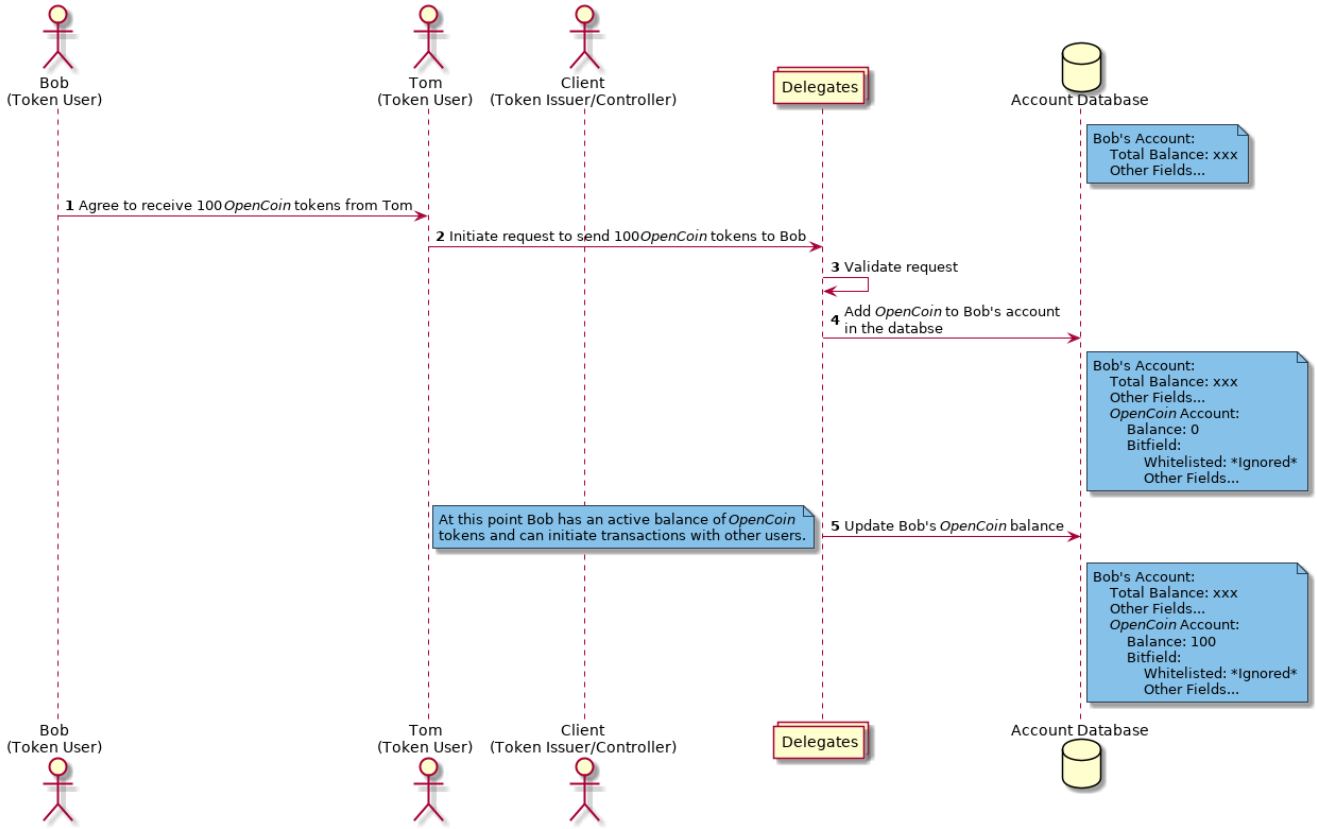
Tokens that use the whitelisting feature require that before receiving any amount of the token, users are first 'whitelisted' by a token controller, after which the user's account will able to receive and send tokens. To save space, rather than maintain a centralized list to store the address of every account that is whitelsited for a particular token, we leverage the token entry added to individual accounts to indicate the account's status.

Whitelisting Process



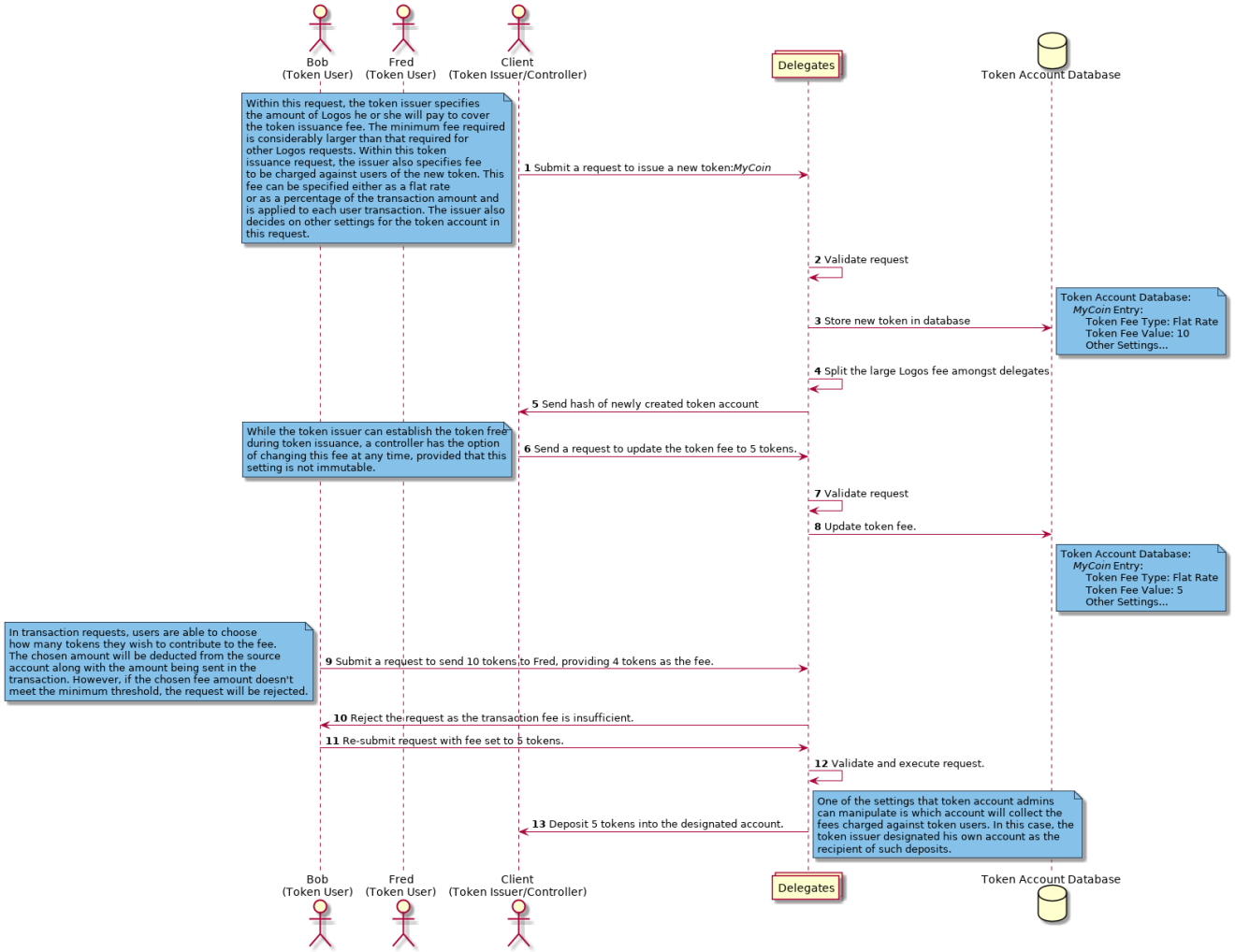
Whitelisting Disabled

Tokens that do not use the whitelisting feature provide no barrier to entry, and any account can receive tokens at will.



Fees

Issueing a new token on the Logos Network requires the issuer to pay a considerable fee (higher than the minimum fee for native send transactions) in Logos. The exact amount required is a configuration parameter determined by the system and as with all Logos fees, the sum will be split among the delegates. Once a token has been issued, all requests involving it are subject to the standard Logos Network fee which is paid in Logos. Requests to send tokens between users are also subject to an additional fee designated by the token's adminitrators (Token Issuer/Controllers), which is paid in tokens. For administrative requests submitted by the controllers, the Logos Network fees are deducted from a balance of Logos owned by the token account. For user requests, the Logos and token fees are deducted from the source account's respective balances, and are respectively distributed amongst the delegates and deposited in an account determined by the token admins.



Token Settings

Each token created on the Logos Network has a list of settings that describe the operations that can be performed on it. The values for these settings are determined by the token issuer when the token is first created, and can be subsequently modified by token controllers with sufficient privilages.

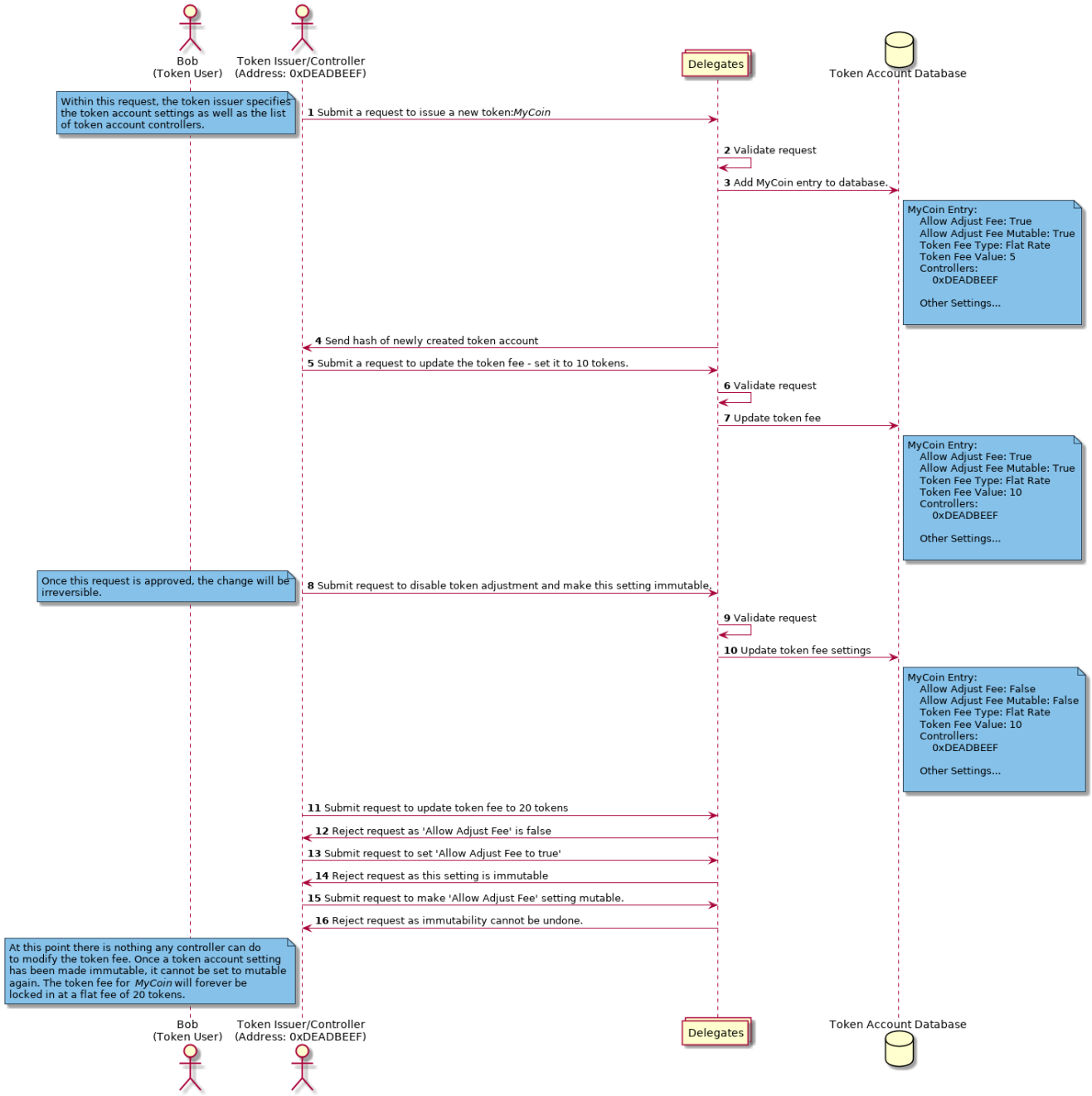
Setting	Description
Allow Additional Tokens	This determines whether the initial supply of tokens can ever be increased.
Allow Additional Tokens Mutable	*
Allow Revoke	This determines whether we can revoke tokens from user account
Allow Revoke Mutable	*

Setting	Description
Allow Freeze	This determines whether we can freeze user accounts.
Allow Freeze Mutable	*
Allow Adjust Fee	This determines whether we can change the transaction fee rate.
Allow Adjust Fee Mutable	*
Need Whitelist	This determines whether accounts need to be whitelisted before receiving tokens.
Need Whitelist Mutable	*

* Mutability setting descriptions are omitted as these settings just determine whether the corresponding setting can be modified. Once a setting is marked as immutable (the mutability setting is set to false), it cannot be changed.

Mutability

A request to modify a token account parameter must be allowed by the corresponding token account setting. Immutable settings cannot be changed at all.



Controllers

A controller is an entity that has administrative privileges for modifying the settings governing a particular token. Token controllers can be appointed by the token issuer when a token is first created, or by a current controller once the token is already live. In order to modify a token account setting, a controller must submit (to a delegate) a request that expresses the desired change. Once this request is approved via Axios Consensus, the change will take affect on the Logos Network. The changes that a controller can make to a token account are dictated by its set of priveleages.

Controller Privilages

Privilege	Description
Allow Additional Tokens	Whether the controller can modify the corresponding token account setting.
Allow Additional Tokens Mutable	"
Allow Revoke	"
Allow Revoke Mutable	"
Allow Freeze	"
Allow Freeze Mutable	"
Allow Adjust Fee	"
Allow Adjust Fee Mutable	"
Need Whitelist	"
Need Whitelist Mutable	"
Promote Controller	Whether the controller can: • Promote another account, making it a controller. • Assign privileges to new or existing controllers. **Note:** A controller with this privilege is similar to a root user on a Linux system as the controller can assign/revoke any privelage to/from any account.
Issue Additional Token	Whether the controller can issue additional tokens.
Revoke	Whether the controller can revoke tokens from an account.
Freeze	Whether the controller can freeze an account.
Adjust Fee	Whether the controller can change the transaction fee rate.
Whitelist	Whether the controller can whitelist an account.
Burn	Whether the controller can burn tokens from the token account balance.
Withdraw Balance	Whether the controller can withdraw tokens from the token account balance.

Privilege	Description
Withdraw Fee	Whether the controller can withdraw tokens from the token account transaction fee pool.

Databases

The Logos Network token platform leverages several databases for storing various information concerning tokens created on the platform:

- **Token Account DB:** This database will be used to store token accounts. Token accounts stored in the database contain their list of settings with current values, as well as the list of token controllers and their respective privileges.
- **Freezelist/Whitelist DB:** Both the freezelist and whitelist are implemented as partially distributed lists - untethered accounts will be stored in a centralized list, while tethered accounts do not need to be stored centrally, since the entries stored in these accounts can be used to demarcate the account's status. The centralized component for both lists will be implemented using a single database. The keys for this database are formed by hashing the token ID with the user's account address and the values for this database are bitfields that indicate this account's freeze and whitelist status for the corresponding token.
- **Account DB:** The account database is used by the base Logos Network and is also leveraged by the token platform, which stores token entries inside of user accounts. Each token entry contains that user's balance of tokens among other useful information.

➔ Next to **Echo**