Overview

This module was taken from Osmosis chain codebase (commit:https://github.com/osmosis-labs/osmosis/commit/2f9c773ab6f10e0b4ea0b2a29264b389a376a9a1). The reason of this action was to adopt module and tests to our codebase and update it to Cosmos SDK 0.47 because it was not possible to import it without code modification that was made by Osmosis team to the original Cosmos SDK. These changes made it not possible (without deep modifications of the whole code) to import module to our code.

Abstract

The tokenfactory module allows any account to create a new token with the namefactory/{creator address}/{subdenom} . Because tokens are namespaced by creator address, this allows token minting to be permissionless, due to not needing to resolve name collisions. A single account can create multiple denoms, by providing a unique subdenom for each created denom. Once a denom is created, the original creator is given "admin" privileges over the asset. This allows them to:

- Mint their denom to any account
- Burn their denom from any account
- Create a transfer of their denom between any two accounts
- Change the admin In the future, more admin capabilities may be
- added. Admins can choose to share admin privileges with other
- accounts using the authz module. TheChangeAdmin
- functionality,
- allows changing the master admin account, or even setting it to""
- , meaning no account has admin privileges of the asset.

Bank hooks

Token factory supports better integration with contracts using bank hooks.

Token factory is integrated with Before Send bank hooks,TrackBeforeSend andBlockBeforeSend . Both hooks gets called whenever a bank send takes place, the difference between two hooks is thatTrackBeforeSend would not error andBlockBeforeSend errors. Due to this differenceTrackBeforeSend is useful for cases when a contract needs to track specific send actions of the token factory denom, whilstBlockBeforeSend would be more useful for situations when we want to block specific sends using contracts.

Each Token Factory denom allows the registration of one contract address. This contract is sudo-called every time the aforementioned bank hooks are activated.

Contracts are able to integrate with these hooks by implementingBlockBeforeSend andTrackBeforeSend message as the following example:

[entry_point]

pub

fn

sudo (deps :

DepsMut , env :

Env , msg :

SudoMsg)

->

StdResult < Response

{ match

& msg { SudoMsg :: BlockBeforeSend

{ from , to , amount }

=>

{ Ok (Response :: new ( ) . add_attributes (vec! [("hook" ,

"block") , ("from" , from) , ("to" , to) , ("amount" ,

& amount . to_string ( ))])) } , SudoMsg :: TrackBeforeSend

{ from , to , amount }

=>

{ Ok (Response :: new ( ) . add_attributes (vec! [("hook" ,

"track") , ("from" , from) , ("to" , to) , ("amount" ,

& amount . to_string ( ))])) } } } Note that sinceTrackBeforeSend hook can also be triggered upon module to module send (which is not gas metered), we internally gas meterTrackBeforeSend with a gas limit of 100_000.

Expectations from the chain

The chain's bech32 prefix for addresses can be at most 16 characters long.

This comes from denoms having a 128 byte maximum length, enforced from the SDK, and us setting longest_subdenom to be 44 bytes. A token factory token's denom is:factory/{creator address}/{subdenom} Splitting up into sub-components, this has:

- len(factory) = 7
- 2 * len("/") = 2
- len(longest_subdenom)
- len(creator_address) = len(bech32(longest_addr_length, chain_addr_prefix))
- .
- Longest addr length at the moment is32 bytes
- .
- Due to SDK error correction settings, this meanslen(bech32(32, chain_addr_prefix)) = len(chain_addr_prefix) + 1 + 58
- .
- Adding this all, we have a total length constraint of128 = 7 + 2 + len(longest_subdenom) + len(longest_chain_addr_prefix) + 1 + 58
- .
- Thereforelen(longest_subdenom) + len(longest_chain_addr_prefix) = 128 - (7 + 2 + 1 + 58) = 60
- .

The choice between how we standardized the split these 60 bytes between maxes from longest_subdenom and longest_chain_addr_prefix is somewhat arbitrary. Considerations going into this:

- PerBIP-0173
- the technically longest HRP for a 32 byte address ('data field') is 31 bytes. (Comes from encode(data) = 59 bytes, and max length = 90 bytes)
- subdenom should be at least 32 bytes so hashes can go into it
- longer subdenoms are very helpful for creating human readable denoms
- chain addresses should prefer being smaller. The longest HRP in cosmos to date is 11 bytes. (persistence
-)

For explicitness, its currently set tolen(longest_subdenom) = 44 andlen(longest_chain_addr_prefix) = 16 .

Please note, if the SDK increases the maximum length of a denom from 128 bytes, these caps should increase. So please don't make code rely on these max lengths for parsing. Previous Overview Next Messages