OASIS PRO
TOKENIZATION

OASIS PRO MARKETS

(TECHNICAL ARCHITECTURE PROVENANCE - SMART CONTRACT)

TABLE OF
CONTENTS

OASIS PRO

Smart Contracts

There are multiple smart contracts present for tokenizing the asset on provenance i.e. execute.rs, init.rs, migrate.rs, mod.rs, query.rs, enum.rs, helper.rs for helper functions etc. which play important role in tokenizing assets. The library used to create token i.e. marker with denom and supply and all the agents and limit is provwasm_std. The Provenance metadata module is used to store and manage additional structured information associated with transactions, accounts, and other entities on the blockchain. In tokenizing Assets on Provenance Blockchain the metadata module is used to asset information on chain and then using the marker contract to tokenize that asset. We are Also using P8E Engine which allows for the execution of privacy-preserving smart contracts, facilitating secure and confidential computations on the Provenance blockchain.

Execute

The Execute has all the functionality from creating token to minting, burning, transferring tokens etc. It also contains support of Multi-sig for mint and burn of tokens. This contracts will be used by the Admin and Agents of the contracts.

Init

The Init contains the logic for instantiating the contract on the chain. This will be used by the deployer of the contract i.e. admin

Query

The Query contains the necessary functions which will be used to query the different states of the contract i.e. balances,agents,multi-sig info etc.

Msg

The Msg contains the various execute and query endpoint messages which will be used by execute, query and init .

States

The States contains all the data mapping used by the contract to read, write and update and store the data of the contract.

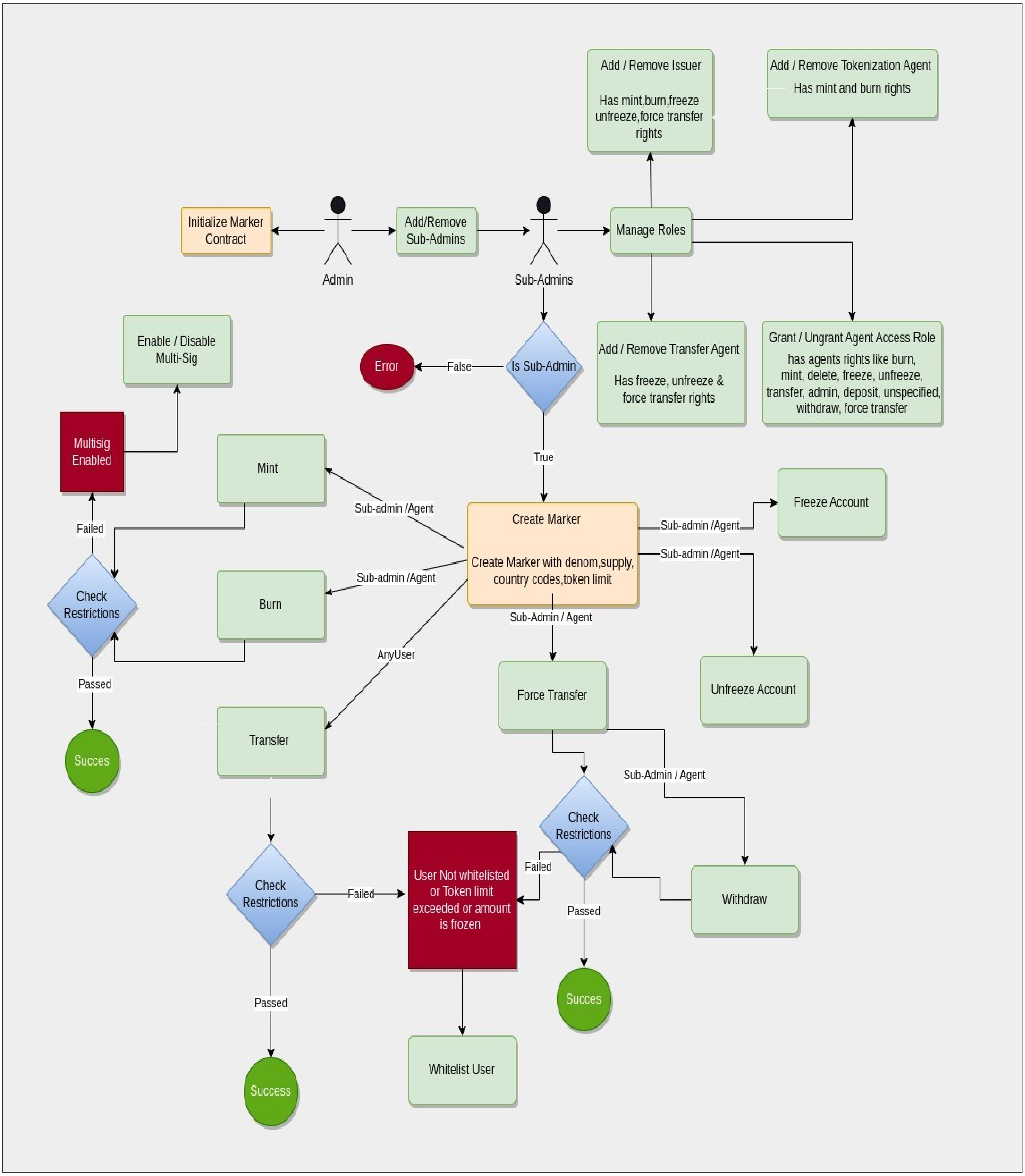
Helper

The Helper contains all the helper functions used by the contract which are being used in every functions so that the same logic does not have to be repeated again and again.

Struct

The Struct contains various struct used by the contract and functions which act as msg for the contract functions.

Flow


Add / Remove Issuer
Has mint,burn,freeze unfreeze,force transfer rights
Add / Remove Tokenization Agent
Has mint and burn rights
Initialize Marker Contract
Admin
Add/Remove Sub-Admins
Sub-Admins
Manage Roles
Enable / Disable Multi-Sig
Error
False
Is Sub-Admin
Add / Remove Transfer Agent
Has freeze, unfreeze & force transfer rights
Grant / Ungrant Agent Access Role
has agents rights like burn, mint, delete, freeze, unfreeze, transfer, admin, deposit, unspecified, withdraw, force transfer
Multisig Enabled
Mint
True
Sub-admin /Agent
Create Marker
Create Marker with denom,supply, country codes,token limit
Freeze Account
Sub-admin /Agent
Sub-admin /Agent
Failed
Check Restrictions
Burn
Sub-admin /Agent
Sub-Admin / Agent
AnyUser
Passed
Force Transfer
Unfreeze Account
Succes
Transfer
Sub-Admin / Agent
Check Restrictions
Failed
User Not whitelisted or Token limit exceeded or amount is frozen
Check Restrictions
Failed
Withdraw
Passed
Succes
Passed
Success
Whitelist User

Token Contract and Functions

The custom_marker contract provide several .rs files which have several functions and events to tokenize assets into tokens and many more. Below is a guide on how to use these functions

Initializing

To initialize the contract from an admin account, i.e., deployer will deploy the contract and he will be registered as admin of the contract who can add sub-admins for the contract. Call the init function in ***init.rs*** :

```
 #[entry_point]
pub fn instantiate(
deps: DepsMut<ProvenanceQuery>,
 _env: Env,
 info: MessageInfo,
_msg: InitMsg,
          ) -> Result<Response<ProvenanceMsg>, ContractError> {
```

This function takes the following parameters:

- **deps:DepsMut:** is a utility type for communicating with the outer world - it allows querying and updating the contract state, querying other contracts state, and gives access to an API object with a couple of helper functions for dealing with CW addresses.
- **_env: Env:** is an object representing the blockchains state when executing the message - the chain height and id, current timestamp, and the called contract address.
- **_info:MessageInfo:** contains metainformation about the message which triggered an execution - an address that sends the message, and chain native tokens sent with the message.
- **_msg:InitMsg:** is the message triggering execution itself - for now, it is InitMsg type that represents Struct in ***struct.rs***, but the type of this argument can be anything that is deserializable, and we can pass more complex types here

The Deps, Env and MessageInfo will be used by every function in the contract so these parameters will not be discussed further.

The function will Dispatch messages to the name module handler and emit an event.

Adding and removing Sub-admin

To add or remove sub-admins, call the ***ManageRoles*** function in ***execute.rs***:

```
pub fn try_manage_roles(
deps: DepsMut<ProvenanceQuery>,
info: MessageInfo,
denom: String,
roles: Vec<Role>,
```

) -> Result<Response<ProvenanceMsg>, ContractError> {

This function takes the following parameters:

- **denom:** The denom of the marker
- **roles:** The vector of role enum type which contains, admin,sub-admin,agents etc.

These functions will response or error based upon the execution of the function. This function will return error ***ContractError::NotSubAdmin*** if the signer is not the subadmin.

Creating Marker

To create token, call the **Create** function in ***execute.rs*** :

fn try_create(

deps: DepsMut<ProvenanceQuery>,

info: MessageInfo,

contract_address: Addr,

params: CreateMarkerParams,

-> Result<Response<ProvenanceMsg>, ContractError> {

This function takes the following parameters:

- **contract_address:** The address of contract
- **params:** It takes ***CreateMarkerParams*** struct which is defined in ***struct.rs*** which contain necessary information of the marker like denom, supply, agents, token limit etc.

This function return Ok response or error based upon the execution of the function and will create marker with provided denom

The ***issuer , tokenization_agent*** and ***transfer_agent*** are the accounts who have different types of access in our contract.

The possible errors to come :

- ***ContractError::NotSubAdmin*** The signer is not sub-admin. To resolve this error, perform ***add_remove_sub_admins***

Burning tokens and Minting tokens

To burn token and mint, call ***burn and mint*** in ***execute.rs:***

fn try_mint_to(

deps: DepsMut<ProvenanceQuery>,

mint_to_params: Vec<MintBurnParams>,

sender: Addr,

) -> Result<Response<ProvenanceMsg>, ContractError> {

fn try_burn_from(

deps: DepsMut<ProvenanceQuery>,

burn_to_params: Vec<MintBurnParams>,

contract_address: Addr,

sender: Addr,

) -> Result<Response<ProvenanceMsg>, ContractError> {

These functions take the following parameters:

- **mint_to_params:** The vector of MintBurnParams defined in struct.rs which contain neccsesary information required to mint
- **burn_to_params:** The vector of MintBurnParams defined in struct.rs which contain neccsesary information required to burn
- **sender:** The address of sender
- **contract_address**: The address of the contract

The possible errors to come :

- **ContractError::Unauthorized :** Multisig is enabled or don't have proper rights of issuer,Tokenization agents or mint and burn rights.

Transfer Tokens

To transfer tokens, call ***transfer*** in ***execute.rs:***

fn try_transfer(

deps: DepsMut<ProvenanceQuery>,

amount: Uint128,

denom: String,

to: Addr,

from: Addr,

) -> Result<Response<ProvenanceMsg>, ContractError> {

This function takes the following parameters:

- **from:** The from address
- **denom:** The denom of the marker
- **to:** The receiver address
- **amount:** The amount to transfer

The possible errors to come :

- **ContractError::CountryCodeAuthorizationFailed :** The account country code is not added so account can't do transaction
- **ContractError::Unauthorized :** The account is blacklisted
- **ContractError::TokenLimitExceeded :** The amount to transfer has exceeded the token limit set during create token

Force Transfer Token

To force transfer token, call ***force_transfer*** in ***oasis_coin.move:***

```
fn try_force_transfer(
deps: DepsMut<ProvenanceQuery>,
denom: String,
params: Vec<ForceTransferParams>,
sender: Addr,
   ) -> Result<Response<ProvenanceMsg>, ContractError> {
```

<u>This function takes the following parameters:</u>

- **denom:** The denom of marker
- **params:** The vector of **ForceTransferParams** defined in **struct.rs**
- **sender:** The sender address

<u>The possible errors to come :</u>

- **ContractError::NotAnIssuer :** The signer does not have rights to perform this action. The signer should be issuer
- **ContractError::NotATransferAgent :** The signer does not have rights to perform this action. The signer should be transfer_agent
- **ContractError::CountryCodeAuthorizationFailed :** The account country code is not added so account can't do transaction
- **ContractError::Unauthorized :** The account is blacklisted
- **ContractError::TokenLimitExceeded :** The amount to transfer has exceeded the token limit set during create token

Freeze and Unfreeze Account

To freeze and unfreeze account call, **F*reeze*** in ***execute.rs:***

```
fn try_update_freezelist(
deps: DepsMut<ProvenanceQuery>,
sender: Addr,
denom: String,
update_type: UpdateType<Vec<Addr>>,
    ) -> Result<Response<ProvenanceMsg>, ContractError> {
```

<u>These functions take the following parameters:</u>

- **denom:** The denom of marker
- **update_type:** The **UpdateType** enum defined in **enum.rs** to add or remove to freeze, unfreeze accounts

• **sender:** The sender address

The possible errors to come :

• **ContractError::Unauthorized :** The account is blacklisted or does not have rights of issuer, Transfer Agent or freeze, unfreeze rights.

Enable Multi-sig

To enable multi-sig, call ***Enable_Multisig*** in ***oasis_coin.move:***

fn try_enable_multisig(

deps: DepsMut<ProvenanceQuery>,

info: MessageInfo,

params: MultisigParams,

) -> Result<Response<ProvenanceMsg>, ContractError> {

This function takes the following parameters:

• **params:** The **MultisigParams** struct defined in **struct.rs**

The possible errors to come :

• **ContractError::Unauthorized :** The Multi-sig is already enabled

• ***ContractError::NotSubAdmin:*** The signer is not sub-admin. To resolve this error, perform ***add_remove_sub_admins***

Manage Proposals

To mange proposals, call ***ManageRequests*** in ***execute.rs:***

fn try_manage_requests(

deps: DepsMut<ProvenanceQuery>,

info: MessageInfo,

env: Env,

request: Requests,

) -> Result<Response<ProvenanceMsg>, ContractError> {

This function takes the following parameters:

• **requests:** The **Requests** enum defined in **enum.rs** which contains various enum and structs to manage proposals

The possible errors to come :

• **ContractError::Unauthorized :** The Multi-sig is not enabled or not the signer who can approve the proposal or have not approved the proposal yet

- **ContractError::Cancelled :** The proposal is cancelled
- **ContractError::ProposalExpired :** The propsal is expired
- **ContractError::NotProposer :** The creator of the proposal is invalid or does not match
- **ContractError::InvalidProposalId :** The propsal is invalid

Other Contract Functions

Add Agents and Roles

To add or remove issuer, Tokenization and transfer agents, call **ManageRoles** in ***execute.rs:***

pub fn try_manage_roles(
deps: DepsMut<ProvenanceQuery>,
info: MessageInfo,
denom: String,
roles: Vec<Role>,
) -> Result<Response<ProvenanceMsg>, ContractError> {

<u>These functions take the following parameters:</u>

- **denom:** The denom of the marker
- **roles:** The vector of **Roles** enum defined in **enum.rs**

<u>The possible errors to come :</u>

- **ContractError::Unauthorized :** The signer is not sub-admin
- **ContractError::AlreadyAdded :** The address is already added
- **ContractError::NotFound :** The address trying to remove is not found

White-list account

To white-list or black-list account, call **Whitelist** in ***execute.rs:***

fn try_update_whitelist(
deps: DepsMut<ProvenanceQuery>,
lists: Vec<WhiteListParams>,

```
sender: Addr,
    ) -> Result<Response<ProvenanceMsg>, ContractError> {
```

These functions take the following parameters:

- **lists:** The vector of **WhiteListParams** struct defined in **struct.rs**
- **sender:** The sender address

The possible errors to come :

- **ContractError::Unauthorized :** The signer is not sub-admin
- **ContractError::NotATransferAgent** : The signer does not have rights to perform this action. The signer should be transfer_agent
- **ContractError::CountryCodeAlreadyExists** : The country code already exist
- **ContractError::NotFound**: The address not found

Add or remove Country codes

To add or remove country codes, call **UpdateCountryCode** in ***execute.rs:***

```
fn try_update_country_code(
deps: DepsMut<ProvenanceQuery>,
update_type: UpdateType<u8>,
denom: String,
sender: Addr,
   ) -> Result<Response<ProvenanceMsg>, ContractError> {
```

These functions take the following parameters:

- **update_type:** The **UpdateType** enum defined in **enum.rs** to add or remove
- **denom:** The denom of marker
- **sender:** The sender address

The possible errors to come :

- **ContractError::Unauthorized :** The signer is not sub-admin
- **ContractError::MissingDenomConfig:** The denom config is missing

Partial freeze and unfreeze balances

To partial freeze and unfreeze balances, call **PartialFreeze** in ***execute.rs:***

```
fn try_partial_freeze(
deps: DepsMut<ProvenanceQuery>,
sender: Addr,
denom: String,
```

params: Vec<PartialFreezeParams>,

) -> Result<Response<ProvenanceMsg>, ContractError> {

This function takes the following parameters:

- **sender:** The account of signer
- **denom:** The denom of marker
- **params :** The vector of **PartialFreezeParams** struct defined in **struct.js**

The possible errors to come :

- **ContractError::NotAnIssuer :** The signer does not have rights to perform this action. The signer should be issuer
- **ContractError::NotATransferAgent:** The signer does not have rights to perform this action. The signer should be Transfer Agent
- **ContractError::NoFreezeAccess:** The signer does not have rights to perform this action. The signer should have Freeze Access

Update Token Limit

To update the token limit set during the creation of token, call **UpdateTokenLimit** in ***execute.rs :***

fn try_update_token_limit(

deps: DepsMut<ProvenanceQuery>,

denom: String,

limit: Uint128,

sender: Addr,

) -> Result<Response<ProvenanceMsg>, ContractError> {

This function takes the following parameters:

- **denom:** The denom of marker
- **sender:** The sender address
- **limit:** The new token limit

The possible errors to come :

- **ContractError::Unauthorized :** The signer is not sub-admin
- **ContractError::MissingDenomConfig:** The denom config is missing