Creators Documentations¶

Key Terminologies¶

- Bonding Tokens
-
    - The assets on the AssetMantle chain can be created in a permissionless environment. The nominal gas fees for on-chain transactions might create an attack vector to possibly unnecessarily mint transactions/spam the assets or related transactions on-chain. One such mechanism is to tackle the bonding fees for the purchases. Bonding is the process of locking the tokens till they are at the end of their lifecycle and burnt. The bonding fees per asset would bedirectly proportional to the number of on-chain metadata properties
- .
- Burn
-
    - An NFT being stored on-chain can't be 'deleted', however it can be 'burned'. Due to the immutability nature of blockchain, the asset, once minted, will exist forever in transaction history. Thus, the Burn mechanism enables you to burn an NFT and will be deleted from the chain state. Utilizing the MantleModules, creators can also set up the burn block height as the parameter, which enables them to burn an asset once the particular height is reached. You can find more information about the usage of the samehere

Types of NFTs and advanced capabilities using MantleModules¶

- Fractionalization
- Fractional NFTs(F-NFTs), in simple terms, divide the ownership of an NFT into smaller fractions. Thus, enabling the possibility for several users to own a fraction of the NFT/Collection
- Use - Cases
-
    - Metaverse
-
    - : Buy Virtual Land and similar digital assets within the virtual world
-
    - Real Estate
-
    - : Multiple parties sharing the rights/ownership of the property.
- Composable/Decomposable
- Composable NFTs
- : Composable NFTs initially introduced inERC998
- , allow the creation of a composed version of more than one asset and add more utilities on top of the existing assets.
- Use-Cases
-
    - P2E & Gaming
-
    - : Combine the rare traits of two different skins in the game to make your character stronger and rarer, and mint the composed assets as NFTs.
-
    - Metaverse
-
    - :For all the in-app characters, you can buy a hat, boot, and jacket for your avatar and mint it as a unique asset.
- Decomposable NFTs
- : Decompose the greatest assets or their associated properties into multiple individual assets.
- 1/1s
- 1/1, often framed as one-of-one, is an NFT that has been issued as a single, unique edition. One-of-ones are more scarce and provide a more valuable set of utilities due to the singular ownership of the asset
- 1-of-many
- Multiple copies of the single edition exist wherein the user can own a single collection piece with a fixed set of utilities/properties.

API List and Definition¶

The following API lists provides an high-level overview about the transactions possible utilizing the MantleModules. For more info, please refer toMantleModules

Transaction / Identity¶

APIs pertaining to manipulating the 'identity', a DID based entity used to represent identity of user.

defineIdentity.define

¶

used to define an identity entity. Defines a classification(schema) of type identity and initiates the transaction.

Function Signature

define: (address: string, chain_id: string, mnemonic: string, fromID: string, mutableTraits: string, immutableTraits: any, mutableMetaTraits: any, immutableMetaTraits: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string) => Promise;

---

defineIdentity.createIdentityDefineMsg

¶

used to define an identity entity. Only defines and outputs a message object.

Function Signature

createIdentityDefineMsg: (address: string, chain_id: string, fromID: string, mutableTraits: string, immutableTraits: any, mutableMetaTraits: any, immutableMetaTraits: any, feesAmount: any, feesToken: any, gas: any, memo: string) => Promise;

---

issueIdentity.issue

¶

used to issue an identity entity. Defines a message and initiates the transaction.

Function Signature

issue: (address: string, chain_id: string, mnemonic: any, to: any, fromID: string, classificationID: any, mutableProperties: string, immutableProperties: any, mutableMetaProperties: any, immutableMetaProperties: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string) => Promise;

---

issueIdentity.createIdentityIssueMsg

¶

used to issue an identity entity. Only defines and outputs a message object.

Function Signature

createIdentityIssueMsg: (address: string, chain_id: string, to: any, fromID: string, classificationID: any, mutableProperties: string, immutableProperties: any, mutableMetaProperties: any, immutableMetaProperties: any, feesAmount: any, feesToken: any, gas: any, memo: string) => Promise;

---

nubIdentity.nub

¶

used to create a nub ID, which is the simplest but complete and valid unit of identification. Defines a message and initiates the transaction.

Function Signature

nub: (address: string, chain_id: string, mnemonic: any, nubID: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string) => Promise;

---

nubIdentity.createIdentityNubMsg

¶

used to create a nub ID, which is the simplest but complete and valid unit of identification. Only defines and outputs a message object.

Function Signature

createIdentityNubMsg: (address: string, chain_id: string, nubID: any, feesAmount: any, feesToken: any, gas: any, memo: string) => Promise;

---

provisionIdentity.provision

¶

used to add a address which is used to control / operate an identity. a nub ID, which is the simplest but complete and valid unit of identification. Defines a message and initiates the transaction.

Function Signature

provision: (address: string, chain_id: string, mnemonic: any, identityID: any, to: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string) => Promise;

---

provisionIdentity.createIdentityProvisionMsg

¶

used to create a provision address which is used to control / operate an identity. Only defines and outputs a message object

Function Signature

createIdentityProvisionMsg: (address: string, chain_id: string, identityID: any, to: any, feesAmount: any, feesToken: any, gas: any, memo: string) => Promise;

---

queryIdentities.queryIdentity

¶

used to query the list of all identities. Defines a message and initiates the transaction.

Function Signature

queryIdentity: () => Promise;

---

queryIdentities.queryIdentityWithID

¶

used to query the list of all identities pertaining to a certain ID argument passed. Defines a message and initiates the transaction.

Function Signature

queryIdentityWithID: (id: any) => Promise;

transaction / assets¶

APIs pertaining to manipulating the 'assets', a DID based entity used to represent NFTs.

burnAsset.burn

¶

used to burn an Asset. Defines a message and initiates the transaction.

Function Signature

burn: (address: string, chain_id: string, mnemonic: string, fromID: string, assetID: string, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

burnAsset.createAssetBurnMsg

¶

used to burn an Asset. Only defines and outputs a message object

Function Signature

createAssetBurnMsg: (address: string, chain_id: string, fromID: string, assetID: string, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

defineAsset.define

¶

used to define the schema of an Asset. Defines a message and initiates the transaction.

Function Signature

define: (address: string, chain_id: string, mnemonic: string, fromID: string, mutableTraits: string, immutableTraits: any, mutableMetaTraits: any, immutableMetaTraits: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

defineAsset.createAssetDefineMsg

¶

used to define the schema of an Asset. Only defines and outputs a message object

Function Signature

createAssetDefineMsg: (address: string, chain_id: string, fromID: string, mutableTraits: string, immutableTraits: any, mutableMetaTraits: any, immutableMetaTraits: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

mintAsset.mint

¶

used to mint an asset using the defined schema. Defines a message and initiates the transaction.

Function Signature

mint: (address: string, chain_id: string, mnemonic: any, toID: any, fromID: string, classificationID: any, mutableProperties: string, immutableProperties: any, mutableMetaProperties: any, immutableMetaProperties: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

mintAsset.createAssetMintMsg

¶

used to mint an asset using the defined schema. Only defines and outputs a message object.

Function Signature

createAssetMintMsg: (address: string, chain_id: string, toID: any, fromID: string, classificationID: any, mutableProperties: string, immutableProperties: any, mutableMetaProperties: any, immutableMetaProperties: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

mutateAsset.mutate

¶

used to change the mutable properties of an Asset. These specific properties must be already defined as mutable in the classification of asset. Defines a message and initiates the transaction.

Function Signature

mutate: (address: string, chain_id: string, mnemonic: any, fromID: string, assetID: any, mutableProperties: string, mutableMetaProperties: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

mutateAsset.createAssetMutateMsg

¶

used to change the mutable properties of an Asset. These specific properties must be already defined as mutable in the schema of asset. Only defines and outputs a message object.

Function Signature

createAssetMutateMsg: (address: string, chain_id: string, fromID: string, assetID: any, mutableProperties: string, mutableMetaProperties: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

queryAssets.queryAsset

¶

queries asset perstating to specific assets of group assetID. These specific properties must be already defined as mutable in the schema of asset. Defines a message and initiates the transaction.

Function Signature

queryAsset: () => Promise;

---

queryAssets.queryAssetWithID

¶

used to query the mutable properties of an Asset with Id. These specific properties must be already defined as mutable in the schema of asset. Only defines and outputs a message object.

Function Signature

queryAssetWithID: (id: any) => Promise;

transaction / classification¶

APIs pertaining to manipulating the Classifications or schemas of entities created

cls.queryClassification

¶

used to query the details of all Classifications IDs.

Function Signature

queryClassification: () => Promise;

---

cls.queryClassificationWithID

¶

used to query the details of Classification IDs pertaining to the ID argument provided in the function

Function Signature

queryClassificationWithID: (id: any) => Promise;

---

transaction / maintainer¶

APIs pertaining to managing the maintainers and access control lists of various entities

deputizeMaintainer.deputize

¶

used to provide or revoke access controls to new maintainer roles. Defines a message and initiates the transaction.

Function Signature

deputize: (address: string, chain_id: string, mnemonic: string, identityID: string, clsID: string, toID: string, maintainedTraits: string, addMaintainer: any, removeMaintainer: any, mutateMaintainer: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

deputizeMaintainer.createDeputizeMsg

¶

used to provide or revoke access controls to new maintainer roles. Only defines and outputs a message object.

Function Signature

createDeputizeMsg: (address: string, chain_id: string, identityID: string, clsID: string, toID: string, maintainedTraits: string, addMaintainer: any, removeMaintainer: any, mutateMaintainer: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

queryMaintainer.queryMaintainerWithID

¶

used to query the details of maintianer pertaining to the ID argument provided in the function

Function Signature

queryMaintainerWithID: (id: any) => Promise;

transaction / order¶

APIs pertaining to manipulating the 'order', a DID based entity used to represent orders created to perform a transfer of value between splites of 'assets' or 'wrapped coins'.

defineOrder.define

¶

used to define a schema for an order type. Defines a message and initiates the transaction.

Function Signature

define: (address: string, chain_id: string, mnemonic: string, fromID: string, mutableTraits: string, immutableTraits: any, mutableMetaTraits: any, immutableMetaTraits: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

defineOrder.createOrderDefineMsg

¶

used to define a schema for an order type. Only defines and outputs a message object.

Function Signature

createOrderDefineMsg: (address: string, chain_id: string, fromID: string, mutableTraits: string, immutableTraits: any, mutableMetaTraits: any, immutableMetaTraits: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

makeOrder.make

¶

used to issue an order pertaining to a specific schema. Defines a message and initiates the transaction.

Function Signature

make: (address: string, chain_id: string, mnemonic: any, fromID: string, classificationID: any, makerOwnableID: string, takerOwnableID: string, expiresIn: any, makerOwnableSplit: any, mutableProperties: string, immutableProperties: any, mutableMetaProperties: any, immutableMetaProperties: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

makeOrder.createOrderMakeMsg

¶

used to issue an order pertaining to a specific schema. Only defines and outputs a message object.

Function Signature

createOrderMakeMsg: (address: string, chain_id: string, fromID: string, classificationID: any, makerOwnableID: string, takerOwnableID: string, expiresIn: any, makerOwnableSplit: any, mutableProperties: string, immutableProperties: any, mutableMetaProperties: any, immutableMetaProperties: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

takeOrder.take

¶

used to put an take order on order book pertaining to a specific order ID. Defines a message and initiates the transaction.

Function Signature

take: (address: string, chain_id: string, mnemonic: any, fromID: string, takerOwnableSplit: any, orderID: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

takeOrder.createOrderTakeMsg

¶

used to accept an issued order pertaining to a specific order ID. Only defines and outputs a message object.

Function Signature

createOrderTakeMsg: (address: string, chain_id: string, fromID: string, takerOwnableSplit: any, orderID: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

cancelOrder.cancel

¶

used to cancel an issued order pertaining to a specific order ID. Defines a message and initiates the transaction.

Function Signature

cancel: (address: string, chain_id: string, mnemonic: string, fromID: string, orderID: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

cancelOrder.createOrderCancelMsg

¶

used to cancel an issued order pertaining to a specific order ID. Only defines and outputs a message object.

Function Signature

createOrderCancelMsg: (address: string, chain_id: string, fromID: string, orderID: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

queryOrders.queryOrder

¶

used to query for the list of all orders available

Function Signature

queryOrder: () => Promise;

---

queryOrders.queryOrderWithID

¶

used to query for a specifc order pertaining to an order ID

Function Signature

queryOrderWithID: (id: any) => Promise;

transaction / meta¶

APIs pertaining to querying a certain meta property across the entities

queryMeta.queryMetaWithID

¶

used to query metaProperty with assetId

Function Signature

queryMetaWithID: (id: any) => Promise;

---

revealMeta.reveal

¶

used to reveal the metaProperties for the asset. Defines a message and initiates the transaction.

Function Signature

reveal: (address: string, chain_id: string, mnemonic: any, metaFact: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

revealMeta.createMetaRevealMsg

¶

used to reveal the hashed mataProperties. Only defines and outputs a message object.

Function Signature

createMetaRevealMsg: (address: string, chain_id: string, metaFact: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

transaction / splits¶

APIs pertaining to creating and manipulating fractionalizing of NFTs and their correlation with FTs

querySplits.querySplitsWithID

¶

used to query a split (fractional NFT) using a specific ID

Function Signature

querySplitsWithID: (id: any) => Promise;

---

sendSplits.send

¶

used to send a split from one identity to another. Defines a message and initiates the transaction.

Function Signature

send: (address: string, chain_id: string, mnemonic: string, fromID: string, toID: string, ownableID: string, split: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

sendSplits.createSplitsSendMsg

¶

used to send a split from one identity to another. Only defines and outputs a message object.

Function Signature

createSplitsSendMsg: (address: string, chain_id: string, fromID: string, toID: string, ownableID: string, split: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

wrapSplits.wrap

¶

used to wrap a coin into a split, which is then used for transacting (transfer of value) in AssetMantle's NFT Economy. Defines a message and initiates the transaction.

Function Signature

wrap: (address: string, chain_id: string, mnemonic: string, fromID: string, coins: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

wrapSplits.createSplitsWrapMsg

¶

used to wrap a coin into a split, which is then used for transacting (transfer of value) in AssetMantle's NFT Economy. Only defines and outputs a message object.

Function Signature

createSplitsWrapMsg: (address: string, chain_id: string, fromID: string, coins: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

unwrapsplits.unwrap

¶

used to unwrap a split back into a coin, which is then used for transacting (transfer of value) in Cosmos Economy and beyond. Defines a message and initiates the transaction.

Function Signature

unwrap: (address: string, chain_id: string, mnemonic: string, fromID: string, ownableID: string, split: any, feesAmount: any, feesToken: any, gas: any, mode: any, memo: string,) => Promise;

---

unwrapsplits.createSplitsUnwrapMsg

¶

used to unwrap a split back into a coin, which is then used for transacting (transfer of value) in Cosmos Economy and beyond. Only defines and outputs a message object.

Function Signature

createSplitsUnwrapMsg: (address: string, chain_id: string, fromID: string, ownableID: string, split: any, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

Transaction / accounts¶

APIs pertaining to creating an account in AssetMantle chain for storing and transacting on coins

createAccount.create

¶

used to create and initiate an account in the AssetMantle chain

Function Signature

create: (address: string, chain_id: string, mnemonic: string, name: string, denom: string, amount: any, gas: any, memo: string,) => Promise;

---

recoverAccount.recover

¶

used to recover an existing account using a mnemonic

Function Signature

recover: (mnemonic: string, name: string) => Promise;

transaction / bank¶

APIs pertaining to transfer of value of coins in the AssetMantle chain and beyond.

bank.sendCoin

¶

APIs pertaining to transfer of value of coins in the AssetMantle chain and beyond.

Function Signature

create: (address: string, chain_id: string, mnemonic: string, name: string, denom: string, amount: any, gas: any, memo: string,) => Promise;

---

bank.createSendCoinMsg

¶

APIs pertaining to transfer of value of coins in the AssetMantle chain and beyond. Only defines and outputs a message object.

Function Signature

createSendCoinMsg: (from_address: string, chain_id: string, to_address: string, denom: string, amount: string, feesAmount: any, feesToken: any, gas: any, memo: string,) => Promise;

---

utilities / keys¶

APIs pertaining to creation of wallets and keystores in the AssetMantle chain.

createWallet

¶

used to create a wallet using a mnemonic and a bip39 passphrase.

Function Signature

createWallet: (mnemonic: string, bip39Passphrase: string,) => Promise<{ address: string; mnemonic: string; }>;

---

createRandomWallet

¶

used to create a wallet using a bip39 passphrase, while the mnemonic is generated automatically.

Function Signature

createRandomWallet: (bip39Passphrase: string) => Promise<{ address: string; mnemonic: string; }>;

---

getWallet

¶

used to get the object pertaining to already created wallet, pertaining to a certain mnemonic and bip39 passphrase

Function Signature

getWallet: (mnemonic: string, bip39Passphrase: string) => Promise;

---

createStore

¶

used to create a encrypted keystore using a specified mnemonic and password

Function Signature

createStore: (mnemonic: string, password: string,) => { Response: any; error: any; };

---

decryptStore

¶

used to decrypt / open and encrypted keystore file, by specifying the file path and a password to decrypt.

Function Signature

decryptStore: (fileData: any, password: string,) => { mnemonic: any; };
