Decentralized Autonomous Organizations

Decentralized Autonomous Organizations (DAOs) are self-organized groups that form around common purposes. Membership, decision making, and funding are coordinated by publicly voting on proposals through a smart contract.

In contrast withFT andNFT , DAO contract's are not standardized. Because of this, in this page we will use as reference theAstra dao contract . The main concepts covered here should easily generalizable to other DAO implementations.

Create a DAO

The simplest way to create and interact with a DAO is to go through theAstraDAO UI .

You can also create a DAO by interacting with thesputnik-dao contract.

- ⚛ Component
- WebApp
- CLI
- Contract

const args =

{ config :

{ name :

"Primitives" , purpose :

"Building primitives on NEAR" , metadata :

"" } , policy :

["bob.near"] } ; Near . call ("sputnik-dao.near" , "create" , { name :

"primitives" , args :

Buffer . from (JSON . stringify (args)) . toString ("base64") , } , 300000000000000 , 600000000000000000000000) ; note The full list of roles and permissions you can findhere . import

{

Wallet

}

from

'./near-wallet' ;

const

DAO_FACTORY_CONTRACT_ADDRESS

=

"sputnik-dao.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

DAO_FACTORY_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

'create' , args :

{ name :

"primitives" , args :

btoa ({ config :

{ name :

"Primitives" , purpose :

"Building primitives on NEAR" , metadata :

"" } , policy :

["bob.near"] }) , } , contractId :

DAO_FACTORY_CONTRACT_ADDRESS , gas :

300000000000000 , deposit :

60000000000000000000000 }) ; note The full list of roles and permissions you can findhere . TheWallet object comes from ourquickstart template export COUNCIL='["bob.near"]' export ARGS=echo '{"config": {"name": "Primitives", "purpose": "Building primitives on NEAR", "metadata":""}, "policy": 'COUNCIL'}' | base64

near call sputnikv2.testnet create "{\"name\": \"primitives\", \"args\": \"ARGS\"}" --accountId bob.near --amount 6 --gas 150000000000000 note The full list of roles and permissions you can findhere . // Validator interface, for cross-contract calls

[ext_contract(ext_dao_factory_contract)]

trait

ExternalDaoFactoryContract

{ fn

create (& mut

self , name :

AccountId , args :

Base64VecU8)

->

Promise ; }

// Implement the contract structure

[near_bindgen]

impl

Contract

{

[payable]

pub

fn

create_dao (& mut

self , name :

AccountId , args :

Base64VecU8)

->

Promise

{ let promise =

ext_dao_factory_contract :: ext (self . dao_factory_contract . clone ( )) . with_attached_deposit (env :: attached_deposit ( )) . with_static_gas (Gas (30 * TGAS)) . create (name , args) ;

return promise . then (

// Create a promise to callback query_greeting_callback Self :: ext (env :: current_account_id ( )) . with_static_gas (Gas (50 * TGAS)) . external_common_callback ( )) }

[private]

// Public - but only callable by env::current_account_id() pub

fn

external_common_callback (& self ,

[callback_result]

call_result :

Result < ( ) ,

PromiseError

)

{ // Check if the promise succeeded if call_result . is_err ( )

{ log! ("There was an error contacting external contract") } } }

Voting policy

Currently, DAOs support two different types ofvoting policies :TokenWeight , andRoleWeight .

When the vote policy isTokenWeight , the council votes usingtokens . The weigh of a vote is the proportion of tokens used for voting over the token's total supply.

When the vote policy isRoleWeight(role) , the vote weigh is computed as "one over the total number of people with the role".

Details Voting Threshold Both voting policies further include athreshold for passing a proposal, which can be a ratio or a fixed number.

The ratio indicates that you need a proportion of people/tokens to approve the proposal (e.g. half the people need to vote, and to vote positively). A fixed number indicated that you need a specific number of votes/tokens to pass the proposal (e.g. 3 people/tokens are enough to approve the proposal).

List of DAOs

Query the list of DAOs existing in Sputnik Dao.

- ⚛ Component
- WebApp
- CLI

const result =

Near . view ("sputnik-dao.near" ,

"get_dao_list") ; Example response ['ref-finance.sputnik-dao.near' 'gaming-dao.sputnik-dao.near' , ...] import

{

Wallet

}

from

'./near-wallet' ;

const

DAO_FACTORY_CONTRACT_ADDRESS

=

"sputnik-dao.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

DAO_FACTORY_CONTRACT_ADDRESS

}) ;

await wallet . viewMethod ({ method :

'get_dao_list' , args :

{ } , contractId :

DAO_FACTORY_CONTRACT_ADDRESS }) ; TheWallet object comes from our quickstart template near view sputnik-dao.near get_dao_list '{}' Example response ['ref-finance.sputnik-dao.near' 'gaming-dao.sputnik-dao.near', ...]

Query Existing Proposals

These snippets will enable you to query the proposals existing in a particular DAO.

- ⚛ Component
- WebApp
- CLI

const result =

Near . view ("nearweek-news-contribution.sputnik-dao.near" , "get_proposals" , {

from_index :

9262 ,

limit :

2

}) ; Example response [{ id :

9262 , proposer :

'pasternag.near' , description :

'NEAR, a top non-EVM blockchain, has gone live on Router’s Testnet Mandara. With Router Nitro, our flagship dApp, users in the NEAR ecosystem can now transfer test tokens to and from NEAR onto other supported chains. https://twitter.com/routerprotocol/status/1727732303491961232' , kind :

{ Transfer :

{ token_id :

'' , receiver_id :

'pasternag.near' , amount :

'500000000000000000000000' , msg :

null } } , status :

'Approved' , vote_counts :

{

council :

[

1 ,

0 ,

0

]

} , votes :

{

'brzk-93444.near' :

'Approve'

} , submission_time :

'1700828277659425683' } , { id :

9263 , proposer :

'fittedn.near' , description :

'How to deploy BOS componenthttps://twitter.com/BitkubAcademy/status/1728003163318563025?t=PiN6pwS380T1N4JuQXSONA&s=19' , kind :

{ Transfer :

{ token_id :

'' , receiver_id :

'fittedn.near' , amount :

'500000000000000000000000' , msg :

null } } , status :

'InProgress' , vote_counts :

{

'Whitelisted Members' :

[

1 ,

0 ,

0

]

} , votes :

{

'trendheo.near' :

'Approve'

} , submission_time :

'1700832601849419123' }] import

{

Wallet

}

from

'./near-wallet' ;

const

DAO_CONTRACT_ADDRESS

=

"nearweek-news-contribution.sputnik-dao.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

DAO_CONTRACT_ADDRESS

}) ;

await wallet . viewMethod ({ method :

'get_proposals' , args :

{

from_index :

9262 ,

limit :

2

} , contractId :

DAO_CONTRACT_ADDRESS }) ; TheWallet object comes from ourquickstart template near view nearweek-news-contribution.sputnik-dao.near get_proposals '{"from_index": 9262, "limit": 2}' Example response [{ id: 9262, proposer: 'pasternag.near', description: 'NEAR, a top non-EVM blockchain, has gone live on Router's Testnet Mandara. With Router Nitro, our flagship dApp, users in the NEAR ecosystem can now transfer test tokens to and from NEAR onto other supported chains. https://twitter.com/routerprotocol/status/1727732303491961232', kind: { Transfer: { token_id: '', receiver_id: 'pasternag.near', amount: '500000000000000000000000', msg: null } }, status: 'Approved', vote_counts: { council: [1, 0, 0] }, votes: { 'brzk-93444.near': 'Approve' }, submission_time: '1700828277659425683' }, { id: 9263, proposer: 'fittedn.near', description: 'How to deploy BOS componenthttps://twitter.com/BitkubAcademy/status/1728003163318563025?t=PiN6pwS380T1N4JuQXSONA&s=19', kind: { Transfer: { token_id: '', receiver_id: 'fittedn.near', amount: '500000000000000000000000', msg: null } }, status: 'InProgress', vote_counts: { 'Whitelisted Members': [1, 0, 0] }, votes: { 'trendheo.near': 'Approve' }, submission_time: '1700832601849419123' }]

Create proposal

Create a proposal so other users can vote in favor or against it.

- ⚛ Component
- WebApp
- CLI
- Contract

Near . call ("primitives.sputnik-dao.near" , "add_proposal" , { proposal :

{ description :

"My first proposalhttps://docs.near.org/" , kind :

{ Transfer :

{ token_id :

"" , receiver_id :

"bob.near" , amount :

"100000000000000000000000000" , } , } , } , } , 300000000000000 , 100000000000000000000000) ; import

{

Wallet

}

from

'./near-wallet' ;

const

DAO_CONTRACT_ADDRESS

=

"primitives.sputnik-dao.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

DAO_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

'add_proposal' , args :

{ proposal :

{ description :

"My first proposalhttps://docs.near.org/" , kind :

{ Transfer :

{ token_id :

"" , receiver_id :

"bob.near" , amount :

"100000000000000000000000000" , } , } , } , } , contractId :

DAO_CONTRACT_ADDRESS , gas :

300000000000000 , deposit :

100000000000000000000000 }) ; TheWallet object comes from ourquickstart template near call primitives.sputnik-dao.near add_proposal '{"proposal": {"description": "My first proposalhttps://docs.near.org/", "kind": { "Transfer": {"token_id": "", "receiver_id": "bob.near", "amount": "100000000000000000000000000"}}}}' --deposit 0.1 --gas 300000000000000 --accountId bob.near // Account ID that represents a token in near-sdk v3 // Need to keep it around for backward compatibility pub

type

OldAccountId

=

String ;

// How the voting policy votes get weighted.

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone, PartialEq)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Debug))]

[serde(crate =

"near_sdk::serde")] pub

enum

WeightKind

{ // Using token amounts and total delegated at the moment. TokenWeight , // Weight of the group role. Roles that don't have scoped group are not supported. RoleWeight , }

// Direct weight or ratio to total weight, used for the voting policy

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Debug, PartialEq))]

[serde(crate =

"near_sdk::serde")]

[serde(untagged)]

pub

enum

WeightOrRatio

{ Weight (U128) , Ratio (u64 ,

u64) , }

// Defines configuration of the vote

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Debug, PartialEq))]

[serde(crate =

"near_sdk::serde")] pub

struct

VotePolicy

{ // Kind of weight to use for votes. pub weight_kind :

WeightKind , // Minimum number required for vote to finalize. // If weight kind is TokenWeight - this is minimum number of tokens required. // This allows to avoid situation where the number of staked tokens from total supply is too small. // If RoleWeight - this is minimum number of votes. // This allows to avoid situation where the role is got too small but policy kept at 1/2, for example. pub quorum :

U128 , // How many votes to pass this vote. pub threshold :

WeightOrRatio , }

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Debug, PartialEq))]

[serde(crate =

"near_sdk::serde")] pub

enum

RoleKind

{ // Matches everyone, who is not matched by other roles. Everyone , // Member greater or equal than given balance. Can use 1 as non-zero balance. Member (U128) , // Set of accounts. Group (HashSet < AccountId

) , }

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Debug, PartialEq))]

[serde(crate =

"near_sdk::serde")] pub

struct

RolePermission

{ // Name of the role to display to the user. pub name :

String , // Kind of the role: defines which users this permissions apply. pub kind :

RoleKind , // Set of actions on which proposals that this role is allowed to execute. // pub permissions :

HashSet < String

, // For each proposal kind, defines voting policy. pub vote_policy :

HashMap < String ,

VotePolicy

, }

// Defines voting / decision making policy of this DAO

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Debug, PartialEq))]

[serde(crate =

"near_sdk::serde")] pub

struct

Policy

{ // List of roles and permissions for them in the current policy. pub roles :

Vec < RolePermission

, // Default vote policy. Used when given proposal kind doesn't have special policy. pub default_vote_policy :

VotePolicy , // Proposal bond. pub proposal_bond :

U128 , // Expiration period for proposals. pub proposal_period :

U64 , // Bond for claiming a bounty. pub bounty_bond :

U128 , // Period in which giving up on bounty is not punished. pub bounty_forgiveness_period :

U64 , }

// Versioned policy

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Debug, PartialEq))]

[serde(crate =

"near_sdk::serde" , untagged)] pub

enum

VersionedPolicy

{ // Default policy with given accounts as council. Default (Vec < AccountId

) , Current (Policy) , }

// Function call arguments

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Clone, Debug))]

[serde(crate =

"near_sdk::serde")] pub

struct

ActionCall

{ method_name :

String , args :

Base64VecU8 , deposit :

U128 , gas :

U64 , }

// Bounty information.

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Debug))]

[serde(crate =

"near_sdk::serde")] pub

struct

Bounty

{ /// Description of the bounty. pub description :

String , /// Token the bounty will be paid out. /// Can be "" for NEAR or a valid account id. pub token :

OldAccountId , /// Amount to be paid out. pub amount :

U128 , /// How many times this bounty can be done. pub times :

u32 , /// Max deadline from claim that can be spend on this bounty. pub max_deadline :

U64 , }

// Info about factory that deployed this contract and if auto-update is allowed

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Clone, Debug))]

[serde(crate =

"near_sdk::serde")] pub

struct

FactoryInfo

{ pub factory_id :

AccountId , pub auto_update :

bool , }

// Function call arguments

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Clone, Debug))]

[serde(crate =

"near_sdk::serde")] pub

struct

PolicyParameters

{ pub proposal_bond :

Option < U128

, pub proposal_period :

Option < U64

, pub bounty_bond :

Option < U128

, pub bounty_forgiveness_period :

Option < U64

, }

// Votes recorded in the proposal

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone, Debug)]

[serde(crate =

"near_sdk::serde")] pub

enum

Vote

{ Approve

=

0x0 , Reject

=

0x1 , Remove

=

0x2 , }

// Configuration of the DAO

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone, Debug)]

[serde(crate =

"near_sdk::serde")] pub

struct

Config

{ // Name of the DAO. pub name :

String , // Purpose of this DAO. pub purpose :

String , // Generic metadata. Can be used by specific UI to store additional data. // This is not used by anything in the contract. pub metadata :

Base64VecU8 , }

// Kinds of proposals, doing different action

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize)]

[cfg_attr(not(target_arch =

"wasm32"), derive(Clone, Debug))]

[serde(crate =

"near_sdk::serde")] pub

enum

ProposalKind

{ // Change the DAO config. ChangeConfig

{ config :

Config

} , // Change the full policy. ChangePolicy

{ policy :

VersionedPolicy

} , // Add member to given role in the policy. This is short cut to updating the whole policy. AddMemberToRole

{ member_id :

AccountId , role :

String

} , // Remove member to given role in the policy. This is short cut to updating the whole policy. RemoveMemberFromRole

{ member_id :

AccountId , role :

String

} , // Calls receiver_id with list of method names in a single promise. // Allows this contract to execute any arbitrary set of actions in other contracts. FunctionCall

{ receiver_id :

AccountId , actions :

Vec < ActionCall

, } , // Upgrade this contract with given hash from blob store. UpgradeSelf

{ hash :

Base58CryptoHash

} , // Upgrade another contract, by calling method with the code from given hash from blob store. UpgradeRemote

{ receiver_id :

AccountId , method_name :

String , hash :

Base58CryptoHash , } , // Transfers given amount of token_id from this DAO to receiver_id. // If msg is not None, calls ft_transfer_call with given msg. Fails if this base token. // For ft_transfer and ft_transfer_call memo is the description of the proposal. Transfer

{ // Can be "" for NEAR or a valid account id. token_id :

OldAccountId , receiver_id :

AccountId , amount :

U128 , msg :

Option < String

, } , // Sets staking contract. Can only be proposed if staking contract is not set yet. SetStakingContract

{ staking_id :

AccountId

} , // Add new bounty. AddBounty

{ bounty :

Bounty

} , // Indicates that given bounty is done by given user. BountyDone

{ bounty_id :

u64 , receiver_id :

AccountId , } , // Just a signaling vote, with no execution. Vote , // Change information about factory and auto update. FactoryInfoUpdate

{ factory_info :

FactoryInfo

} , // Add new role to the policy. If the role already exists, update it. This is short cut to updating the whole policy. ChangePolicyAddOrUpdateRole

{ role :

RolePermission

} , // Remove role from the policy. This is short cut to updating the whole policy. ChangePolicyRemoveRole

{ role :

String

} , // Update the default vote policy from the policy. This is short cut to updating the whole policy. ChangePolicyUpdateDefaultVotePolicy

{ vote_policy :

VotePolicy

} , // Update the parameters from the policy. This is short cut to updating the whole policy. ChangePolicyUpdateParameters

{ parameters :

PolicyParameters

} , }

[derive(Serialize, Deserialize)]

[serde(crate =

"near_sdk::serde")] pub

struct

ProposalInput

{ /// Description of this proposal. pub description :

String , /// Kind of proposal with relevant information. pub kind :

ProposalKind , }

// Validator interface, for cross-contract calls

[ext_contract(ext_dao_contract)]

trait

ExternalDaoContract

{ fn

add_proposal (& mut

self , proposal :

ProposalInput)

->

Promise ; }

// Implement the contract structure

[near_bindgen]

impl

Contract

{

[payable]

pub

fn

create_proposal (& mut

self , proposal :

ProposalInput)

->

Promise

{ let promise =

ext_dao_contract :: ext (self . dao_contract . clone ( )) . with_attached_deposit (env :: attached_deposit ( )) . with_static_gas (Gas (5 * TGAS)) . add_proposal (proposal) ;

return promise . then (

// Create a promise to callback query_greeting_callback Self :: ext (env :: current_account_id ( )) . with_static_gas (Gas (50 * TGAS)) . external_proposal_callback ( )) }

[private]

// Public - but only callable by env::current_account_id() pub

fn

external_proposal_callback (& self ,

[callback_result]

call_result :

Result < u64 ,

PromiseError

)

->

Option < u64

{ if call_result . is_err ( )

{ log! ("There was an error contacting external contract") ; return

None ; }

// Return the proposal id let id = call_result . unwrap ( ) ; return

Some (id) ; } } info By default, onlycouncil members can create proposals.

Vote for proposal

These snippet will enable your users to cast a vote for proposal of a particular DAO.

- ⚛ Component
- WebApp
- CLI
- Contract

Near . call ("primitives.sputnik-dao.near" , "act_proposal" , {

id :

0 ,

action :

"VoteApprove"

} , 300000000000000) ; note Available vote options:VoteApprove ,VoteReject ,VoteRemove . import

{

Wallet

}

from

'./near-wallet' ;

const

DAO_CONTRACT_ADDRESS

=

"primitives.sputnik-dao.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

DAO_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

'act_proposal' , args :

{

id :

0 ,

action :

"VoteApprove"

} , contractId :

DAO_CONTRACT_ADDRESS , gas :

300000000000000 , }) ; note Available vote options:VoteApprove ,VoteReject ,VoteRemove . TheWallet object comes from ourquickstart template near call primitives.sputnik-dao.near act_proposal '{"id": 0, "action": "VoteApprove"}' --gas 300000000000000 --accountId bob.near note Available vote options:VoteApprove ,VoteReject ,VoteRemove . // Set of possible action to take

[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Debug)]

[serde(crate =

"near_sdk::serde")] pub

enum

Action

{ // Action to add proposal. Used internally. AddProposal , // Action to remove given proposal. Used for immediate deletion in special cases. RemoveProposal , // Vote to approve given proposal or bounty. VoteApprove , // Vote to reject given proposal or bounty. VoteReject , // Vote to remove given proposal or bounty (because it's spam). VoteRemove , // Finalize proposal, called when it's expired to return the funds // (or in the future can be used for early proposal closure). Finalize , // Move a proposal to the hub to shift into another DAO. MoveToHub , }

// Validator interface, for cross-contract calls

[ext_contract(ext_dao_contract)]

trait

ExternalDaoContract

{ fn

act_proposal (& mut

self , id :

u64 , action :

Action , memo :

Option < String

)

->

Promise ; }

// Implement the contract structure

[near_bindgen]

impl

Contract

{

[payable]

pub

fn

act_proposal (& mut

self , id :

u64 , action :

Action , memo :

Option < String

)

->

Promise

{ let promise =

ext_dao_contract :: ext (self . dao_contract . clone ( )) . with_attached_deposit (env :: attached_deposit ( )) . with_static_gas (Gas (10 * TGAS)) . act_proposal (id , action , memo) ;

return promise . then (

// Create a promise to callback query_greeting_callback Self :: ext (env :: current_account_id ( )) . external_common_callback ( )) }

[private]

// Public - but only callable by env::current_account_id() pub

fn

external_common_callback (& self ,

[callback_result]

call_result :

Result < ( ) ,

PromiseError

)

{ // Check if the promise succeeded if call_result . is_err ( )

{ log! ("There was an error contacting external contract") } } }

Additional Resources

1. AstroDAO UI
2.
   - the web app built on top of the Sputnik DAO Contract. Allows users to create and manage DAOs.
3. List of DAOs as a NEAR componentEdit this page Last updatedonJan 31, 2024 bygagdiez Was this page helpful? Yes No

Previous Price Oracles Next Decentralized Exchanges (DEX)