Linkdrops

Linkdrops allow users to distribute assets and onboard people to Web3 apps through a simple web link.

They work by storing assets and linkingAccessKeys to them. TheAccessKeys are then distributed to users in the form of web links. These links take users to a website that automatically uses the keys to call theclaim method in thelinkdrop contract.

In order for a contract to be considered a Linkdrop-contract it has to follow theNEP-452 standard . TheNEP-452 explains theminimum interface required to be implemented, as well as the expected functionality.

Keypom

The simplest way to create Linkdrops is by usingKeypom . Keypom is a community project that allows you to create Linkdrops for NEAR tokens and NFTs in a few clicks.

AccessKeys

In order to create any kind of drop, you need to first generate key pairs. You will need to createone key per drop .

- Thelinkdrop
- contract will store thepublic
- part of the key.
- You will give theprivate

- part of the key to the user you want to receive the drop.

- ⚛ Component

- WebApp

- CLI

- Keypom API

const dropsNumber =

"2" ; const keysGeneratorUrl =

"https://keypom.sctuts.com/keypair/" ; const rootEntrophy =

"my-password" ;

//If not provided, the keypair will be completely random. see: https://docs.keypom.xyz/docs/next/keypom-sdk/Core/modules asyncFetch (keysGeneratorUrl + dropsNumber +

"/"

+ rootEntrophy) . then ((res)

=>

{ const keyPairs =

JSON . parse (res . body) ; const pubKeys =

[ ] ; const privKeys =

[ ] ;

keyPairs . forEach ((e)

=>

{ pubKeys . push (e . pub) ; privKeys . push (e . priv) ; }) ;

const obj =

{ publicKeys : pubKeys , privKeys : privKeys , } ;

State . update (obj) ; }) ; * near-api-js * Keypom API

import

{

KeyPair

}

from

'near-api-js' ;

const newKeyPair =

KeyPair . fromRandom ('ed25519') ; newKeyPair . public_key

= newKeyPair . publicKey . toString ( ) ; const state =

{ } ;

const dropsNumber =

"2" ; const keysGeneratorUrl =

"https://keypom.sctuts.com/keypair/" ;

fetch (keysGeneratorUrl + dropsNumber +

"/rootEntrophy") . then ((res)

=>

{ const keyPairs =

JSON . parse (res . body) ; const pubKeys =

[ ] ; const privKeys =

[ ] ;

keyPairs . forEach ((e)

=>

{ pubKeys . push (e . pub) ; privKeys . push (e . priv) ; }) ;

state . publicKeys

= pubKeys ; state . privKeys

= privKeys ; }) ; * Near CLI * Keypom API

This command creates a key pair locally in .near-credentials with an implicit account as the accountId (hash representation of the public key)

near generate-key Example response:

Key pair with ed25519:33Vn9VtNEtWQPPd1f4jf5HzJ5weLcvGHU8oz7o5UnPqy public key for an account "1e5b1346bdb4fc5ccd465f6757a9082a84bcacfd396e7d80b0c726252fe8b3e8"
export NUMBER_OF_DROPS=2

curl https://keypom.sctuts.com/keypairNUMBER_OF_DROPS/rootEntrophy

NEAR Drops

To create a NEAR drop you will ask the contract to create a drop (create_drop), passing the public part of the keys you generated, and how much you want to drop on each key use (deposit_per_use).

The contract will create a drop andreturn the numerical ID that identifies it.

- ⚛ Component
- WebApp
- CLI

const keypomContract =

"v2.keypom.near" ; const dropAmount =

"10000000000000000000000" ;

Near . call ([{ contractName : keypomContract , methodName :

"create_drop" , args :

{ public_keys : state . publicKeys , deposit_per_use : dropAmount , } , deposit :

"23000000000000000000000" ,

// state.publicKeys.length * dropAmount + 3000000000000000000000, gas :

"100000000000000" , } ,]) ; import

{

Wallet

}

from

'./near-wallet' ;

const

KEYPOM_CONTRACT_ADDRESS

=

"v2.keypom.near" ; const

DROP_AMOUNT

=

"10000000000000000000000" ;

// 0.1 NEAR

const wallet =

new

Wallet ({

createAccessKeyFor :

KEYPOM_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

"create_drop" , contractId :

KEYPOM_CONTRACT_ADDRESS , args :

{ public_keys : state . publicKeys , deposit_per_use :

DROP_AMOUNT , } , deposit :

"23000000000000000000000"

// state.publicKeys.length * dropAmount + 3000000000000000000000, gas :

"100000000000000" , }) ; TheWallet object comes from ouquickstart template near call v2.keypom.near create_drop '{"public_keys":, "deposit_per_use": "10000000000000000000000"}' --depositYocto 23000000000000000000000 --gas 100000000000000 --accountId bob.near info To claim the drop, you will need to send the user alink with the private key

NFT Drops

To drop an existing NFT, you will (1) create a drop, and then (2)transfer the NFT to keypom.

1. Creating the Drop

To create an NFT drop, you will call thecreate_drop method, now passing anft argument, which will tell the linkdrop contract to wait for an NFT to be transferred.

The contract will then create a drop andreturn the numerical ID that identifies it.

- ⚛ Component
- WebApp
- CLI

const accountId = context . accountId

?? props . accountId ; const keypomContract =

"v2.keypom.near" ; const nftContract =

"nft.primitives.near" ; const dropAmount =

"10000000000000000000000" ;

Near . call ([{ contractName : keypomContract , methodName :

"create_drop" , args :

{ public_keys : state . publicKeys , deposit_per_use : dropAmount , nft :

{ // Who will be sending the NFTs to the Keypom contract sender_id : accountId , // NFT Contract Id that the tokens will come from contract_id : nftContract , } , } , deposit :

"23000000000000000000000"

// state.publicKeys.length * dropAmount + 3000000000000000000000, gas :

"100000000000000" , } ,]) ; import

{

Wallet

}

from

'./near-wallet' ;

const

KEYPOM_CONTRACT_ADDRESS

=

"v2.keypom.near" ; const

NFT_CONTRACT_ADDRESS

=

"nft.primitives.near" ; const

DROP_AMOUNT

=

"10000000000000000000000" ;

const keypomConnectedWallet =

new

Wallet ({

createAccessKeyFor :

KEYPOM_CONTRACT_ADDRESS

}) ; const nftConnectedWallet =

new

Wallet ({

createAccessKeyFor :

NFT_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

"create_drop" , contractId :

KEYPOM_CONTRACT_ADDRESS , args :

{ public_keys : state . publicKeys , deposit_per_use :

DROP_AMOUNT , nft :

{ // Who will be sending the NFTs to the Keypom contract sender_id : accountId ,

// TODO How to get it // NFT Contract Id that the tokens will come from contract_id :

NFT_CONTRACT_ADDRESS , } , } , deposit :

"23000000000000000000000"

// state.publicKeys.length * dropAmount + 3000000000000000000000, gas :

"100000000000000" , }) ; TheWallet object comes from ourquickstart template near call v2.keypom.near create_drop '{"public_keys":, "deposit_per_use": "10000000000000000000000", "nft": {"sender_id": "bob.near", "contract_id": "nft.primitives.near"}}' --depositYocto 23000000000000000000000 --gas 100000000000000 --accountId bob.near

2. Transferring the NFT

Having the Drop ID, you now need to transfer the NFT to the linkdrop contract, specifying to which drop you want to add it.

- ⚛ Component
- WebApp
- CLI

const nftTokenId =

"1" ;

Near . call ([{ contractName : nftContract , methodName :

'nft_transfer_call' , args :

{ receiver_id : keypomContract , token_id : nftTokenId , msg : dropId . toString ( ) } , deposit :

"1" , gas :

"300000000000000" }]) ; import

{

Wallet

}

from

'./near-wallet' ;

const

KEYPOM_CONTRACT_ADDRESS

=

"v2.keypom.near" ; const

NFT_CONTRACT_ADDRESS

=

"nft.primitives.near" ; const

NFT_TOKEN_ID

=

"1" ; const

DROP_AMOUNT

=

"10000000000000000000000" ;

const nftConnectedWallet =

new

Wallet ({

createAccessKeyFor :

NFT_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

"nft_transfer_call" , contractId :

NFT_CONTRACT_ADDRESS , args :

{ receiver_id : keypomContract , token_id : nftTokenId , msg : dropId . toString ( ) } , deposit :

1 , gas :

"100000000000000" , }) ; TheWallet object comes from ourquickstart template near call nft.primitives.near nft_transfer_call '{"receiver_id": "v2.keypom.near", "token_id":, "msg": }' --depositYocto 1 --gas 100000000000000 --accountId bob.near tip Thelinkdrop contract will validate that you are transferring the NFT to a drop that belongs to you

FT Drops

The process to drop a Fungible Token is very similar to that of creating anNFT drop . You will first create the drop, and then fund it with FTs.

1.Creating a drop

To create a FT drop you will call thecreate_drop method, now passing aftData argument, which will tell the linkdrop contract to wait for a certain amount of FT to be transferred.

The contract will then create a drop andreturn the numerical ID that identifies it.

- ⚛ Component
- WebApp
- CLI

const keypomContract =

"v2.keypom.near" ; const ftContract =

"ft.primitives.near" ; const dropAmount =

"10000000000000000000000" ;

Near . call ([{ contractName : keypomContract , methodName :

"create_drop" , args :

{ public_keys : state . publicKeys , deposit_per_use : dropAmount , ftData :

{ contractId : ftContract , senderId : accountId , // This balance per use is balance of human readable FTs per use. amount :

"1" // Alternatively, you could use absoluteAmount, which is dependant on the decimals value of the FT // ex. if decimals of an ft = 8, then 1 FT token would be absoluteAmount = 100000000 } , } , deposit :

"23000000000000000000000" ,

// state.publicKeys.length * dropAmount + 3000000000000000000000, gas :

"100000000000000" } ,]) ; import

{

Wallet

}

from

'./near-wallet' ;

const

KEYPOM_CONTRACT_ADDRESS

=

"v2.keypom.near" ; const

FT_CONTRACT_ADDRESS

=

"ft.primitives.near" ; const

DROP_AMOUNT

=

"10000000000000000000000" ;

const wallet =

new

Wallet ({

createAccessKeyFor :

KEYPOM_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

"create_drop" , contractId :

KEYPOM_CONTRACT_ADDRESS , args :

{ public_keys : state . publicKeys , deposit_per_use :

DROP_AMOUNT , ftData :

{ contractId :

FT_CONTRACT_ADDRESS , senderId : accountId ,

// TODO How to get account id // This balance per use is balance of human readable FTs per use. amount :

"1" // Alternatively, you could use absoluteAmount, which is dependant on the decimals value of the FT // ex. if decimals of an ft = 8, then 1 FT token would be absoluteAmount = 100000000 } , } , deposit :

"23000000000000000000000"

// state.publicKeys.length * dropAmount + 3000000000000000000000, gas :

"100000000000000" , }) ; TheWallet object comes from ourquickstart template near call v2.keypom.near create_drop '{"public_keys":, "deposit_per_use": "10000000000000000000000", "ftData": {"contractId": "ft.primitives.near","senderId": "bob.near", "amount": "1"}}}' --depositYocto 23000000000000000000000 --gas 100000000000000 --accountId bob.near

2. Transferring FT

Having the Drop ID, you now need to transfer the fungible tokens to the linkdrop contract.

note To transfer FTs to an account, you need to firstregister the receiver account (e.g. the keypom contract) on the FT contract. * ⚛ Component * WebApp *

Near . call ([{ contractName : ftContract , methodName :

"ft_transfer" , args :

{ receiver_id : keypomContract , amount :

"1" , } , deposit :

"1" , gas :

"300000000000000" , } ,]) ; import

{

Wallet

}

from

'./near-wallet' ;

const

KEYPOM_CONTRACT_ADDRESS

=

"v2.keypom.near" ; const

FT_CONTRACT_ADDRESS

=

"ft.primitives.near" ;

const wallet =

new

Wallet ({

createAccessKeyFor :

FT_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

"ft_transfer" , contractId :

FT_CONTRACT_ADDRESS , args :

{ receiver_id :

KEYPOM_CONTRACT_ADDRESS , amount :

"1" } , deposit :

"1" , gas :

"100000000000000" }) ; TheWallet object comes from ourquickstart template near call ft.primitives.near ft_transfer '{"receiver_id": "v2.keypom.near", "amount": "1"}' --depositYocto 1 --gas 100000000000000 --accountId bob.near

Function Call Drop

Linkdrop contracts allow to createfunction call drops. These drops will execute one or more methods on a contract when the user claims the drop.

tip Function call drops can be thought as the abstract version of other drops: you can create a drop that will mint an NFT, register a user in a DAO, or pay for a service. * ⚛ Component * WebApp * CLI

const accountId = context . accountId

?? props . accountId ; const keypomContract =

"v2.keypom.near" ; const nftContract =

"nft.primitives.near" ; const nftTokenId =

"1" ; const dropAmount =

"10000000000000000000000" ;

Near . call ([{ contractName : keypomContract , methodName :

"create_drop" , args :

{ public_keys : state . publicKeys , deposit_per_use : dropAmount , fcData :

{ // 2D array of function calls. In this case, there is 1 function call to make for a key use // By default, if only one array of methods is present, this array of function calls will be used for all key uses methods :

[// Array of functions for Key use 1. [{ receiverId : nftContract , methodName :

"nft_mint" , args :

JSON . stringify ({ // Change this token_id if it already exists -> check explorer transaction token_id : nftTokenId , metadata :

{ title :

"My NFT drop" , description :

"" , media :

"" , } }) , accountIdField :

"receiver_id" , // Attached deposit for when the receiver makes this function call attachedDeposit :

"10000000000000000000000" }]] } } , deposit :

"23000000000000000000000" ,

// state.publicKeys.length * dropAmount + 3000000000000000000000, gas :

"100000000000000" , } ,]) ; import

{

Wallet

}

from

'./near-wallet' ;

const

KEYPOM_CONTRACT_ADDRESS

=

"v2.keypom.near" ; const

NFT_CONTRACT_ADDRESS

=

"nft.primitives.near" ; const

NFT_TOKEN_ID

=

"1" ; const

DROP_AMOUNT

=

"10000000000000000000000" ;

const wallet =

new

Wallet ({

createAccessKeyFor :

DAO_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

"create_drop" , contractId :

KEYPOM_CONTRACT_ADDRESS , args :

{ public_keys : state . publicKeys , deposit_per_use :

DROP_AMOUNT , fcData :

{ // 2D array of function calls. In this case, there is 1 function call to make for a key use // By default, if only one array of methods is present, this array of function calls will be used for all key uses methods :

[// Array of functions for Key use 1. [{ receiverId :

NFT_CONTRACT_ADDRESS , methodName :

"nft_mint" , args :

JSON . stringify ({ // Change this token_id if it already exists -> check explorer transaction token_id :

NFT_TOKEN_ID , metadata :

{ title :

"My NFT drop" , description :

"" , media :

"" , } }) , accountIdField :

"receiver_id" , // Attached deposit for when the receiver makes this function call attachedDeposit :

"10000000000000000000000" }]] } } , deposit :

"23000000000000000000000"

// state.publicKeys.length * dropAmount + 3000000000000000000000, gas :

"100000000000000" , }) ; TheWallet object comes from ouquickstart template near call v2.keypom.near create_drop '{"public_keys": , "deposit_per_use": "10000000000000000000000", "fcData": {"methods": [[{"receiverId": "nft.primitives.near","methodName": "nft_mint","args": {"token_id": "1", "metadata": {"title": "My NFT drop","description": "","media": ""}, "accountIdField": "receiver_id", "attachedDeposit": "10000000000000000000000"}]]}}' --depositYocto 23000000000000000000000 --gas 100000000000000 --accountId bob.near

Building drop links

const

getLinks

=

( )

=>

{ const links =

[ ] ;

// It assumes that private keys have been already stored in State by using State.init() and State.update() method state . privKeys . map ((e , i)

=>

{ const link = "https://app.mynearwallet.com"

+

"/linkdrop/v2.keypom.near/"

+ e ; links . push (link) ; }) ;

return links ; } ; Example response ['https://app.mynearwallet.com/linkdrop/v2.keypom.near/ed25519:2H32THYM8ob336yk81cZUxpidvKi34zLck6a97ypmCY8bbSAuEfrCTu9LWmWGiG9df2C6vkg2FGKGZzY9qE4aEcj' , 'https://app.mynearwallet.com/linkdrop/v2.keypom.near/ed25519:3eoMcqKmmY9Q6qgBy3hZy65HisZ8NXQd9aGGYUGe6RRsmNpGJS5YN64MgZaBVVYJJhbFXhQ2ca3DRRBiKh1rYM48'] note If you didn't save your linkdrop links before closing NEAR App, you can always find them onKeyPom app .

Additional Resources

1. Linkdrop plus
2. allows to create a Simple Drop. Powered byKeyPom
3. Keypom Drop Viewer
4. shows drops created by current logged in user.Edit this page Last updatedonMar 25, 2024 bygagdiez Was this page helpful? Yes No

Previous Non-Fungible Tokens (NFT) Next Price Oracles