Fungible Tokens (FT)

Besides the native NEAR token, NEAR accounts have access to amultitude of tokens to use thoughtout the ecosystem. Moreover, it is even possible for users to create their own fungible tokens.

In contrast with the NEAR native token, fungible token (FT) arenot stored in the user's account. In fact, each FT lives intheir own contract which is in charge of doingbookkeeping . This is, the contract keeps track of how many tokens each user has, and handles transfers internally.

In order for a contract to be considered a FT-contract it has to follow theNEP-141 and NEP-148 standards . TheNEP-141 &NEP-148 standards explain theminimum interface required to be implemented, as well as the expected functionality.

Token Factory

You can create an FT using the community toolToken Farm . Token farm is a token factory, you can interact with it through its graphical interface, or by making calls to its contract.

- ⚛ Component
- WebApp
- CLI

const args =

{ args :

{ owner_id :

"bob.near" , total_supply :

"1000000000" , metadata :

{ spec :

"ft-1.0.0" , name :

"Test Token" , symbol :

"test" , icon :

"" , decimals :

18 , } , } , account_id :

"bob.near" , } ;

Near . call ("tkn.near" ,

"create_token" , args ,

300000000000000 ,

"223483000000000000000000") ; import

{

Wallet

}

from

'./near-wallet' ;

const wallet =

new

Wallet ({ }) ;

const args =

{ args :

{ owner_id :

"bob.near" , total_supply :

"1000000000" , metadata :

{ spec :

"ft-1.0.0" , name :

"Test Token" , symbol :

"test" , icon :

"" , decimals :

18 , } , } , account_id :

"bob.near" , } ;

await wallet . callMethod ({ method :

'create_token' , args , contractId :

"tkn.near" , gas :

300000000000000 , deposit :

"223483000000000000000000" }) ; TheWallet object comes from ourquickstart template near call tkn.near create_token '{"args":{"owner_id": "bob.near","total_supply": "1000000000","metadata":{"spec": "ft-1.0.0","name": "Test Token","symbol": "TTTEST","icon": "","decimals": 18}},"account_id": "bob.near"}' --gas 300000000000000 --depositYocto 223483000000000000000000 --accountId bob.near The FT you create will live in the account.tkn.near (e.g.test.tkn.near).

Deploying Your Own Contract

You can also create a fungible token by deploying and initializing a canonical FT contract .

On initialization you will define the token's metadata such as its name (e.g. Ethereum), symbol (e.g. ETH) and total supply (e.g. 10M). You will also define anowner , which will own the tokenstotal supply .

To initialize a FT contract you will need to deploy it and then call thenew method defining the token's metadata.

near deploy --wasmFile fungible_token.wasm

near call new '{"owner_id": "", "total_supply": "1000000000000000", "metadata": { "spec": "ft-1.0.0", "name": "Example Token Name", "symbol": "EXLT", "decimals": 8 }}' --accountId tip Check theContract Wizard to create a personalized FT contract!.

Querying Metadata

You can query the FT's metadata by calling theft_metadata .

- ⚛ Component
- WebApp
- CLI

const tokenContract =

"token.v2.ref-finance.near" ; const tokenMetadata =

Near . view (tokenContract ,

"ft_metadata" ,

{ }) ; Example response { "spec": "ft-1.0.0", "name": "Ref Finance Token", "symbol": "REF", "icon": "data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' viewBox='16 24 248 248' style='background: %23000'%3E%3Cpath d='M164,164v52h52Zm-45-45,20.4,20.4,20.6-20.6V81H119Zm0,18.39V216h41V137.19l-20.6,20.6ZM166.5,81H164v33.81l26.16-26.17A40.29,40.29,0,0,0,166.5,81ZM72,153.19V216h43V133.4l-11.6-11.61Zm0-

18.38,31.4-31.4L115,115V81H72ZM207,121.5h0a40.29,40.29,0,0,0-7.64-23.66L164,133.19V162h2.5A40.5,40.5,0,0,0,207,121.5Z' fill='%23fff'/%3E%3Cpath d='M189 72l27 27V72h-27z' fill='%2300c08b'/%3E%3C/svg%3E%0A", "reference": null, "reference_hash": null, "decimals": 18 } import

{

Wallet

}

from

'./near-wallet' ;

const

TOKEN_CONTRACT_ADDRESS

=

"token.v2.ref-finance.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

TOKEN_CONTRACT_ADDRESS

}) ;

await wallet . viewMethod ({ method :

'ft_metadata' , args :

{ } , contractId :

TOKEN_CONTRACT_ADDRESS }) ; Example response { "spec": "ft-1.0.0", "name": "Ref Finance Token", "symbol": "REF", "icon": "data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' viewBox='16 24 248 248' style='background: %23000'%3E%3Cpath d='M164,164v52h52Zm-45-45,20.4,20.4,20.6-20.6V81H119Zm0,18.39V216h41V137.19l-20.6,20.6ZM166.5,81H164v33.81l26.16-26.17A40.29,40.29,0,0,0,166.5,81ZM72,153.19V216h43V133.4l-11.6-11.61Zm0-18.38,31.4-31.4L115,115V81H72ZM207,121.5h0a40.29,40.29,0,0,0-7.64-23.66L164,133.19V162h2.5A40.5,40.5,0,0,0,207,121.5Z' fill='%23fff'/%3E%3Cpath d='M189 72l27 27V72h-27z' fill='%2300c08b'/%3E%3C/svg%3E%0A", "reference": null, "reference_hash": null, "decimals": 18 } TheWallet object comes from ourquickstart template near view token.v2.ref-finance.near ft_metadata Example response { spec: "ft-1.0.0", name: "Ref Finance Token", symbol: "REF", icon: "data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' viewBox='16 24 248 248' style='background: %23000'%3E%3Cpath d='M164,164v52h52Zm-45-45,20.4,20.4,20.6-20.6V81H119Zm0,18.39V216h41V137.19l-20.6,20.6ZM166.5,81H164v33.81l26.16-26.17A40.29,40.29,0,0,0,166.5,81ZM72,153.19V216h43V133.4l-11.6-11.61Zm0-18.38,31.4-31.4L115,115V81H72ZM207,121.5h0a40.29,40.29,0,0,0-7.64-23.66L164,133.19V162h2.5A40.5,40.5,0,0,0,207,121.5Z' fill='%23fff'/%3E%3Cpath d='M189 72l27 27V72h-27z' fill='%2300c08b'/%3E%3C/svg%3E%0A", reference: null, reference_hash: null, decimals: 18 }

Checking Balance

To know how many coins a user has you will need to query the methodft_balance_of .

- ⚛ Component
- WebApp
- CLI

info Remember about fungible token precision. You may need this value to show a response of balance requests in an understandable-to-user way in your app. How to get precision value (decimals) you may findabove . const tokenContract =

"token.v2.ref-finance.near" ; const userTokenBalance =

Near . view (tokenContract ,

"ft_balance_of" ,

{ account_id :

"bob.near" , }) ; Example response "3479615037675962643842" info Remember about fungible token precision. You may need this value to show a response of balance requests in an understandable-to-user way in your app. How to get precision value (decimals) you may findabove . import

{

Wallet

}

from

'./near-wallet' ;

const

TOKEN_CONTRACT_ADDRESS

=

"token.v2.ref-finance.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

TOKEN_CONTRACT_ADDRESS

}) ;

await wallet . viewMethod ({ method :

'ft_balance_of' , args :

{ account_id :

'bob.near' } , contractId :

TOKEN_CONTRACT_ADDRESS }) ; Example response "3479615037675962643842" TheWallet object comes from ourquickstart template near view token.v2.ref-finance.near ft_balance_of '{"account_id": "bob.near"}' Example response '37622432282532717742 6'

Registering a User

In order for an user to own and transfer tokens they need to firstregister in the contract. This is done by callingstorage_deposit and attaching 0.00125Ⓝ.

By calling thisstorage_deposit the user can register themselves orregister other users .

- ⚛ Component
- WebApp
- CLI

Near . call (tokenContract , "storage_deposit" , {

account_id :

"alice.near"

} , undefined , 125000000000000000000) ; await wallet . callMethod ({ method :

'storage_deposit' , args :

{ account_id :

'alice.near' , } , contractId :

TOKEN_CONTRACT_ADDRESS , deposit :

125000000000000000000 }) ; TheWallet object comes from ourquickstart template near call token.v2.ref-finance.near storage_deposit '{"account_id": "alice.near"}' --depositYocto 125000000000000000000 --accountId bob.near info You can

make sure a user is registered by callingstorage_balance_of . tip After a user calls thestorage_deposit the FT will appear in their Wallets.

Transferring Tokens

To send FT to another account you will use theft_transfer method, indicating the receiver and the amount of FT you want to send.

- ⚛ Component
- WebApp
- CLI
- Contract

const tokenContract =

"token.v2.ref-finance.near" ; Near . call (tokenContract , "ft_transfer" , { receiver_id :

"alice.near" , amount :

"100000000000000000" , } , undefined , 1) ; import

{

Wallet

}

from

'./near-wallet' ;

const

TOKEN_CONTRACT_ADDRESS

=

"token.v2.ref-finance.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

TOKEN_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

'ft_transfer' , args :

{ receiver_id :

'alice.near' , amount :

'100000000000000000' , } , contractId :

TOKEN_CONTRACT_ADDRESS , deposit :

1 }) ; TheWallet object comes from ouquickstart template near call token.v2.ref-finance.near ft_transfer '{"receiver_id": "alice.near", "amount": "100000000000000000"}' --depositYocto 1 --accountId bob.near

[near_bindgen]

impl

Contract

{

[payable]

pub

fn

send_tokens (& mut

self , receiver_id :

AccountId , amount :

U128)

->

Promise

{ assert_eq! (env :: attached_deposit ( ) ,

1 ,

"Requires attached deposit of exactly 1 yoctoNEAR") ;

let promise =

ext (self . ft_contract . clone ( )) . with_attached_deposit (YOCTO_NEAR) . ft_transfer (receiver_id , amount ,

None) ;

return promise . then (

// Create a promise to callback query_greeting_callback Self :: ext (env :: current_account_id ( )) . with_static_gas (Gas (30 * TGAS)) . external_call_callback ( )) }

[private]

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

fn

external_call_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") ; } } } This snippet assumes that the contract is already holding some FTs and that you want to send them to another account.

Attaching FTs to a Call

Natively, only NEAR tokens (Ⓝ) can be attached to a function calls. However, the FT standard enables to attach fungible tokens in a call by using the FT-contract as intermediary. This means that, instead of you attaching tokens directly to the call, you ask the FT-contract to do both a transfer and a function call in your name.

Let's assume that you need to deposit FTs on Ref Finance.

- ⚛ Component
- WebApp

- CLI
- Contract

const tokenContract =

"token.v2.ref-finance.near" ; const result =

Near . call (tokenContract , "ft_transfer_call" , { receiver_id :

"v2.ref-finance.near" , amount :

"100000000000000000" , msg :

"" , } , 300000000000000 , 1) ; Example response '100000000000000000' import

{

Wallet

}

from

'./near-wallet' ;

const

TOKEN_CONTRACT_ADDRESS

=

"token.v2.ref-finance.near" ; const wallet =

new

Wallet ({

createAccessKeyFor :

TOKEN_CONTRACT_ADDRESS

}) ;

await wallet . callMethod ({ method :

'ft_transfer_call' , args :

{ receiver_id :

"v2.ref-finance.near" , amount :

"100000000000000000" , msg :

"" , } , contractId :

TOKEN_CONTRACT_ADDRESS , gas :

300000000000000 , deposit :

1 }) ; Example response '100000000000000000' TheWallet object comes from ouquickstart template near call token.v2.ref-finance.near ft_transfer_call '{"receiver_id": "v2.ref-finance.near", "amount": "100000000000000000", "msg": ""}' --gas 300000000000000 --depositYocto 1 --accountId bob.near Example response '100000000000000000'

[payable]

pub

fn

call_with_attached_tokens (& mut

self , receiver_id :

AccountId , amount :

U128)

->

Promise

{ assert_eq! (env :: attached_deposit ( ) ,

1 ,

"Requires attached deposit of exactly 1 yoctoNEAR") ;

let promise =

ext (self . ft_contract . clone ( )) . with_static_gas (Gas (150 * TGAS)) . with_attached_deposit (YOCTO_NEAR) . ft_transfer_call (receiver_id , amount ,

None ,

"" . to_string ( )) ;

return promise . then (

// Create a promise to callback query_greeting_callback Self :: ext (env :: current_account_id ( )) . with_static_gas (Gas (100 * TGAS)) . external_call_callback ( )) } How it works:

1. You call ft_transfer_call in the FT contract passing: the receiver, a message, and the amount.
2. The FT contract transfers the amount to the receiver.
3. The FT contract calls receiver.ft_on_transfer(sender, msg, amount)
4. The FT contract handles errors in the ft_resolve_transfer callback.
5. The FT contract returns you how much of the attached amount was actually used.

Handling Deposits (Contract Only)

If you want your contract to handle deposit in FTs you have to implement theft_on_transfer method. When executed, such method will know:

- Which FT was transferred, since it is the predecessor account.
- Who is sending the FT, since it is a parameter
- How many FT were transferred, since it is a parameter
- If there are any parameters encoded as a message

Theft_on_transfer must return how many FT tokens have tobe refunded , so the FT contract gives them back to the sender.

// Implement the contract structure

[near_bindgen]

impl

Contract

{ }

[near_bindgen]

impl

FungibleTokenReceiver

for

Contract

{ // Callback on receiving tokens by this contract. //msg format is either "" for deposit or TokenReceiverMessage. fn

ft_on_transfer (& mut

self , sender_id :

AccountId , amount :

U128 , msg :

String ,)

->

PromiseOrValue < U128

{ let token_in =

env :: predecessor_account_id ( ) ;

assert! (token_in ==

self . ft_contract ,

"{}" ,

"The token is not supported") ; assert! (amount

=

self . price ,

"{}" ,

"The attached amount is not enough") ;

env :: log_str (format! ("Sender id: {:?}" , sender_id) . as_str ( )) ;

if msg . is_empty ( )

{ // Your internal logic here PromiseOrValue :: Value (U128 (0)) }

else

{ let message = serde_json :: from_str :: < TokenReceiverMessage

(& msg) . expect ("WRONG_MSG_FORMAT") ; match message { TokenReceiverMessage :: Action

{ buyer_id , }

=>

{ let buyer_id = buyer_id . map (| x | x . to_string ( )) ; env :: log_str (format! ("Target buyer id: {:?}" , buyer_id) . as_str ( )) ; // Your internal business logic PromiseOrValue :: Value (U128 (0)) } } } } }

Additional Resources

1. NEP-141 and NEP-148 standards
2. FT Event Standards
3. FT reference implementation
4. Fungible Tokens 101
5.
    - a set of tutorials that cover how to create a FT contract using RustEdit this page Last updatedonFeb 9, 2024 bygagdiez Was this page helpful? Yes No

Previous What are Primitives?Next Non-Fungible Tokens (NFT)