

# Non-Fungible Tokens (NFT)

In contrast with fungible tokens, non-fungible tokens (NFT) are unitary and therefore unique. This makes NFTs ideal to represent ownership of assets such as a piece of digital content, or a ticket for an event.

As with fungible tokens, NFTs are not stored in the user's wallet, instead, each NFT lives in an NFT contract. The NFT contract works as a bookkeeper, this is: it is in charge of handling the creation, storage and transfers of NFTs.

In order for a contract to be considered a NFT-contract it has to follow the [NEP-171 and NEP-177 standards](#). The NEP-171 & NEP-177 standards explain the minimum interface required to be implemented, as well as the expected functionality.

NFT & Marketplaces Be mindful of not confusing an NFT with an NFT-marketplace. NFT simply store information (metadata), while NFT-marketplaces are contracts where NFT can be listed and exchanged for a price.

## Community Projects

The easiest way to create and handle NFTs is by using one of the existing community projects.

1. [Paras](#)
2.
  - a classic NFT marketplace. Just login with your NEAR account, create a collection and share the link with your community.
3. [Mintbase](#)
4.
  - a marketplace that allows to create NFT collections, and buy NFTs using credit cards or stablecoins.
5. [Enleap](#)
6.
  - a no-code launchpad for NFTs. Provides NFT minting, staking, whitelist managing, tracking functionality.

## Deploying a NFT Contract


If you want to deploy your own NFT contract, you can create one using our [reference implementation](#)

Simply personalize it and deploy it to your account.

`near deploy --wasmFile contract.wasm --initFunction new tip` Check the [Contract Wizard](#) to create a personalized NFT contract!.

## Minting a NFT

To create a new NFT (a.k.a. minting it) you will call the `thenft_mint` method passing as arguments the metadata that defines the NFT.

-  Component
  - WebApp
  - CLI
  - Contract
- Reference
  - Paras
  - Mintbase

```
const tokenData =
```

```
Near . call ( "nft.primitives.near" , "nft_mint" , { token_id :
```

```
"1" , receiver_id :
```

```
"bob.near" , token_metadata :
```

```
{ title :
```

```
"NFT Primitive Token" , description :
```

```
"Awesome NFT Primitive Token" , media :
```

```
"string" ,
```

[illegible]

$$\} ) ;$$

```
await wallet . callMethod ( { method :
```

```
'nft_batch_mint' , args :
```

```
{ num_to_mint :
```

```
1 , owner_id :
```

```
"bob.near" , metadata :
```

```
{ } , } , contractId :
```

```
CONTRACT_ADDRESS , deposit :
```

```
1000000000000000000000000
```

```
// Depends on your NFT metadata } ) ; note In order to use nft_batch_mint method of Mintbase store contract your account  
have to be a in the contract minters list. tip Check how to do this using Mintbase JS TheWallet object comes from  
our quickstart template * Reference * Paras * Mintbase
```

```
near call nft.primitives.near nft_mint '{"token_id": "1", "receiver_id": "bob.near", "token_metadata": {"title": "NFT Primitive  
Token", "description": "Awesome NFT Primitive Token", "media": "string"}}' --depositYocto 1000000000000000000000000, --  
accountId bob.near near call x.paras.near nft_mint '{"token_series_id": "490641", "receiver_id": "bob.near"}' --depositYocto  
1000000000000000000000000 --accountId bob.near note In order to use nft_mint method of thex.paras.near contract you have  
to be a creator of a particular token series. near call thomasettorreiv.mintbase1.near nft_batch_mint '{"num_to_mint": 1,  
"owner_id": "bob.near", "metadata": {}}' --accountId bob.near --depositYocto 1000000000000000000000000 note In order to  
use nft_batch_mint method of Mintbase store contract your account have to be a in the contract minters list. // Validator  
interface, for cross-contract calls
```

## [ext\_contract(ext\_nft\_contract)]

```
trait
```

```
ExternalNftContract
```

```
{ fn
```

```
nft_mint ( & self , token_series_id :
```

```
String , receiver_id :
```

```
AccountId )
```

```
->
```

```
Promise ; }
```

```
// Implement the contract structure
```

## [near\_bindgen]

```
impl
```

```
Contract
```

```
{
```

## [payable]

```
pub
```

```
fn
```

```
nft_mint ( & mut
```

```
self , token_series_id :
```

```
String , receiver_id :
```

AccountId )

->

Promise

{ let promise =

```
ext_nft_contract :: ext ( self . nft_contract . clone ( ) ) . with_static_gas ( Gas ( 30 * TGAS ) ) . with_attached_deposit ( env :: attached_deposit ( ) ) . nft_mint ( token_series_id , receiver_id ) ;
```

```
return promise . then (
```

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

## [private]

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

```
fn
```

```
nft_mint_callback ( & self ,
```

## [callback\_result]

```
call_result :
```

```
Result < TokenId ,
```

```
PromiseError
```

```
)
```

->

```
Option < TokenId
```

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

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

```
None ; }
```

```
// Return the token data let token_id :
```

```
TokenId
```

```
= call_result . unwrap ( ) ; return
```

Some ( token\_id ) ; } } info See the [metadata standard](#) for the full list of TokenMetadata parameters. warning Values of gas and deposit might vary depending on which NFT contract you are calling.

## Minting Collections

Many times people want to create multiple 100 copies of an NFT (this is called a collection). In such cases, what you actually need to do is to mint 100 different NFTs with the same metadata (but different token-id ).

tip Notice that [minting in Mintbase](#) allows you to pass anum\_to\_mint parameter.

## Royalties

You might have noticed that one of the parameters is a structure called royalties. Royalties enable you to create a list of users that should get paid when the token is sell in a marketplace. For example, ifanna has 5% of royalties, each time the NFT is sell,anna should get a 5% of the selling price.

## Querying NFT data

You can query the NFT's information and metadata by calling thenft\_token .

- \* Component
- WebApp
- CLI
- Contract
- Reference
- Paras
- Mintbase

const tokenData =

Near . view ( "nft.primitives.near" ,

"nft\_token" ,

{ token\_id :

"1" , } ) ; Example response { "token\_id": "1", "owner\_id": "bob.near", "metadata": { "title": "string", // ex. "Arch Nemesis: Mail Carrier" or "Parcel #5055" "description": "string", // free-form description "media": "string", // URL to associated media, preferably to decentralized, content-addressed storage "media\_hash": "string", // Base64-encoded sha256 hash of content referenced by the media field. Required if media is included. "copies": 1, // number of copies of this set of metadata in existence when token was minted. "issued\_at": 1642053411068358156, // When token was issued or minted, Unix epoch in milliseconds "expires\_at": 1642053411168358156, // When token expires, Unix epoch in milliseconds "starts\_at": 1642053411068358156, // When token starts being valid, Unix epoch in milliseconds "updated\_at": 1642053411068358156, // When token was last updated, Unix epoch in milliseconds "extra": "string", // anything extra the NFT wants to store on-chain. Can be stringified JSON. "reference": "string", // URL to an off-chain JSON file with more info. "reference\_hash": "string" // Base64-encoded sha256 hash of JSON from reference field. Required if reference is included. } } const tokenData =

fetch ( "https://api-v2-mainnet.paras.id/token?token\_id=84686:1154" ) ; Example response { "status": 1, "data": { "results": [ { "\_id": "61dfbf27284abc1cc0b87c9d", "contract\_id": "x.paras.near", "token\_id": "84686:1154", "owner\_id": "bob.near", "token\_series\_id": "84686", "edition\_id": "1154", "metadata": { "title": "Tokenfox Silver Coin #1154", "description": "Holding this silver coin in your wallet will bring you health and happiness \uD83D\uDE0A", "media": "bafkreihpapfu7rzsmejjgl2twillge6pbrfmqaahj2wkz6nq55c6trhhtrq", "media\_hash": null, "copies": 4063, "issued\_at": null, "expires\_at": null, "starts\_at": null, "updated\_at": null, "extra": null, "reference": "bafkreib6uj5kxbadfvf6qes5flema7jx6u5dj5zyqcneaoyqqzlm6kpu5a", "reference\_hash": null, "collection": "Tokenfox Collection Cards", "collection\_id": "tokenfox-collection-cards-by-tokenfoxnear", "creator\_id": "tokenfox.near", "blurhash": "U7F~gc00\_3D%00~q4n%M\_39F~RjM{xuWBRj", "score": 0, "mime\_type": "image/png" }, "royalty": { "tokenfox.near": 1000 }, "price": null, "approval\_id": null, "ft\_token\_id": null, "has\_price": null, "is\_creator": true, "total\_likes": 8, "likes": null, "categories": [], "view": 4 } ], "count": 1, "skip": 0, "limit": 10 } } info See the [Paras API documentation](#) for the full list of methods. note Paras API methods returns data from all NFT contracts in NEAR. You might want to pass more parameters like contract\_id or owner\_id to make the response more accurate. const tokenData =

fetch ( "https://graph.mintbase.xyz" ,

{ method :

"POST" , headers :

{ "mb-api-key" :

"anon" , "Content-Type" :

"application/json" , "x-hasura-role" :

"anonymous" , } , body :

JSON . stringify ( { query :

query getToken{ tokens: nft\_tokens( where: { token\_id: { \_eq: "84686:1154" } } ) { tokenId: token\_id ownerId: owner contractId: nft\_contract\_id reference issuedAt: issued\_at copies metadataId: metadata\_id } , } ) , } } ; Example response { "ok": true, "status": 200, "contentType": "application/json", "body": { "data": { "tokens": [ { "tokenId": "84686:1154", "ownerId": "bob.near", "contractId": "x.paras.near", "reference": "bafkreib6uj5kxbadfvf6qes5flema7jx6u5dj5zyqcneaoyqqzlm6kpu5a", "issuedAt": "2022-01-13T05:56:51.068358", "copies": 4063, "metadataId": "x.paras.near:5210047642790498956c9669d6a37b98" } ] } } } note In the future, users may be required to register using an api key. For now, simply passing the value anon form mb-api-key will work. \* Reference \* Paras \* Mintbase

import

{

Wallet

}

from

'./near-wallet' ;

const

CONTRACT\_ADDRESS

=

"nft.primitives.near" ; const wallet =

new

Wallet ( {

createAccessKeyFor :

CONTRACT\_ADDRESS

} ) ;

const response =

await wallet . viewMethod ( { method :

'nft\_token' , args :

{ token\_id :

"1" } } ) ; TheWallet object comes from our [quickstart template](#)

Example response { "token\_id": "1", "owner\_id": "bob.near", "metadata": { "title": "string", // ex. "Arch Nemesis: Mail Carrier" or "Parcel #5055" "description": "string", // free-form description "media": "string", // URL to associated media, preferably to decentralized, content-addressed storage "media\_hash": "string", // Base64-encoded sha256 hash of content referenced by the media field. Required if media is included. "copies": 1, // number of copies of this set of metadata in existence when token was minted. "issued\_at": 1642053411068358156, // When token was issued or minted, Unix epoch in milliseconds "expires\_at": 1642053411168358156, // When token expires, Unix epoch in milliseconds "starts\_at": 1642053411068358156, // When token starts being valid, Unix epoch in milliseconds "updated\_at": 1642053411068358156, // When token was last updated, Unix epoch in milliseconds "extra": "string", // anything extra the NFT wants to store on-chain. Can be stringified JSON. "reference": "string", // URL to an off-chain JSON file with more info. "reference\_hash": "string" // Base64-encoded sha256 hash of JSON from reference field. Required if reference is included. } } const tokenData =

fetch ( "https://api-v2-mainnet.paras.id/token?token\_id=84686:1154" ) ; TheWallet object comes from our [quickstart template](#)

Example response { "status": 1, "data": { "results": [ { "\_id": "61dfbf27284abc1cc0b87c9d", "contract\_id": "x.paras.near", "token\_id": "84686:1154", "owner\_id": "bob.near", "token\_series\_id": "84686", "edition\_id": "1154", "metadata": { "title": "Tokenfox Silver Coin #1154", "description": "Holding this silver coin in your wallet will bring you health and happiness \uD83D\uDE0A", "media": "bafkreihpapfu7rzsmejjgl2twllge6pbrfmqaahj2wkz6nq55c6trhhtrq", "media\_hash": null, "copies": 4063, "issued\_at": null, "expires\_at": null, "starts\_at": null, "updated\_at": null, "extra": null, "reference": "bafkreib6uj5kxbadfvf6qes5flema7jx6u5dj5zyqcneaoyqqzlm6kpu5a", "reference\_hash": null, "collection": "Tokenfox Collection Cards", "collection\_id": "tokenfox-collection-cards-by-tokenfoxnear", "creator\_id": "tokenfox.near", "blurhash": "U7F~gc00\_3D%00~q4n%M\_39F-RjM{xuWBRj", "score": 0, "mime\_type": "image/png" }, "royalty": { "tokenfox.near": 1000 }, "price": null, "approval\_id": null, "ft\_token\_id": null, "has\_price": null, "is\_creator": true, "total\_likes": 8, "likes": null, "categories": [], "view": 4 } ], "count": 1, "skip": 0, "limit": 10 } } info See the [Paras API documentation](#) for the full list of methods. note Paras API methods returns data from all NFT contracts in NEAR. You might want to pass more parameters like contract\_id or owner\_id to make the response more accurate. const tokenData =

fetch ( "https://graph.mintbase.xyz" ,

{ method :

"POST" , headers :

{ "mb-api-key" :

"anon" , "Content-Type" :

"application/json" , "x-hasura-role" :

"anonymous" , } , body :

JSON . stringify ( { query :

query getToken{ tokens: nft\_tokens( where: { token\_id: { \_eq: "84686:1154" } } ) { tokenId: token\_id ownerId: owner contractId: nft\_contract\_id reference issuedAt: issued\_at copies metadataId: metadata\_id } } , } ) , } ) ; TheWallet object comes from our [quickstart template](#)

Example response { "ok": true, "status": 200, "contentType": "application/json", "body": { "data": { "tokens": [ { "tokenId": "84686:1154", "ownerId": "bob.near", "contractId": "x.paras.near", "reference": "bafkreib6uj5kxbadfvf6qes5flema7jx6u5dj5zyqcneaoyqqzlm6kpu5a", "issuedAt": "2022-01-13T05:56:51.068358", "copies": 4063, "metadataId": "x.paras.near:5210047642790498956c9669d6a37b98" } ] } } } note In the future, users may be required to register using an api key. For now, simply passing the valueanon formb-api-key will work. tip Check how to do this also using the [Mintbase JS](#) \* Reference \* Paras \* Mintbase

near view nft.primitives.near nft\_token '{"token\_id": "1"}' Example response { "token\_id": "1", "owner\_id": "bob.near", "metadata": { "title": "string", // ex. "Arch Nemesis: Mail Carrier" or "Parcel #5055" "description": "string", // free-form description "media": "string", // URL to associated media, preferably to decentralized, content-addressed storage "media\_hash": "string", // Base64-encoded sha256 hash of content referenced by the media field. Required if media is included. "copies": 1, // number of copies of this set of metadata in existence when token was minted. "issued\_at": 1642053411068358156, // When token was issued or minted, Unix epoch in milliseconds "expires\_at": 1642053411168358156, // When token expires, Unix epoch in milliseconds "starts\_at": 1642053411068358156, // When token starts being valid, Unix epoch in milliseconds "updated\_at": 1642053411068358156, // When token was last updated, Unix epoch in milliseconds "extra": "string", // anything extra the NFT wants to store on-chain. Can be stringified JSON. "reference": "string", // URL to an off-chain JSON file with more info. "reference\_hash": "string" // Base64-encoded sha256 hash of JSON from reference field. Required if reference is included. } } near view x.paras.near nft\_token '{"token\_id": "84686:1154"}' Example response { "token\_id": "84686:1154", "owner\_id": "bob.near", "metadata": { "title": "Tokenfox Silver Coin #1154", "description": null, "media": "bafkreihpapfu7rzsmejjgl2twwlge6pbrfmqaahj2wkz6nq55c6trhhtrq", "media\_hash": null, "copies": 4063, "issued\_at": "1642053411068358156", "expires\_at": null, "starts\_at": null, "updated\_at": null, "extra": null, "reference": "bafkreib6uj5kxbadfvf6qes5flema7jx6u5dj5zyqcneaoyqqzlm6kpu5a", "reference\_hash": null }, "approved\_account\_ids": {} } near view anthropocene.mintbase1.near nft\_token '{"token\_id": "17960"}' Example response { "token\_id": "17960", "owner\_id": "876f40299dd919f39252863e2136c4e1922cd5f78759215474cbc8f1fc361e14", "approved\_account\_ids": {}, "metadata": { "title": null, "description": null, "media": null, "media\_hash": null, "copies": 1, "issued\_at": null, "expires\_at": null, "starts\_at": null, "updated\_at": null, "extra": null, "reference": "F-30s\_uQ3ZdAHZCIY4DYatDPapalRNLju41RxfMXC24", "reference\_hash": null }, "royalty": { "split\_between": { "seventhage.near": { "numerator": 10000 } }, "percentage": { "numerator": 100 } }, "split\_owners": null, "minter": "anthropocene.seventhage.near", "loan": null, "composeable\_stats": { "local\_depth": 0, "cross\_contract\_children": 0 }, "origin\_key": null } note When someone creates a NFT on Mintbase they need to deploy their own NFT contract using Mintbase factory. Those smart contract are subaccounts of mintbase1.near, e.g.anthropocene.mintbase1.near . // Validator interface, for cross-contract calls

## [ext\_contract(ext\_nft\_contract)]

trait

ExternalNftContract

{ fn

nft\_token ( & self , token\_id :

TokenId )

->

Promise ; }

// Implement the contract structure

## [near\_bindgen]

impl

Contract

{ pub

fn



```

nft_token ( & self , token_id :
TokenId )

->

Promise

{ let promise =

ext_nft_contract :: ext ( self . nft_contract . clone ( ) ) . nft_token ( token_id ) ;

return promise . then (

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

```

## [private]

```

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

fn

nft_token_callback ( & self ,

```

## [callback\_result]

```

call_result :

Result < Token ,

PromiseError

    )

->

Option < Token

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

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

None ; }

// Return the token data let token_data :

Token

= call_result . unwrap ( ) ; return





Some ( token_data ) ; } }

```

## Transferring a NFT

Transferring an NFT can happen in two scenarios: (1) you ask to transfer an NFT, and (2) an [authorized account](#) asks to transfer the NFT.

In both cases, it is necessary to invoke the `nft_transfer` method, indicating the token id, the receiver, and an (optionally) an [approval id](#).

-  Component
-  WebApp
-  CLI
-  Contract
- Reference
- Paras
- Mintbase

```

const tokenData =
Near . call ( "nft.primitives.near" , "nft_transfer" , { token_id :
"1" , receiver_id :
"bob.near" } , undefined , 1 , ) ; const tokenData =
Near . call ( "x.paras.near" , "nft_transfer" , { token_id :
"490641" , receiver_id :
"bob.near" } , undefined , 1 ) ; const tokenData =
Near . call ( "thomasettorreiv.mintbase1.near" , "nft_transfer" , { token_id :
"490641" , receiver_id :
"bob.near" } , undefined , 1 ) ; * Reference * Paras * Mintbase

```

```
import
```

```
{
```

```
Wallet
```

```
}
```

```
from
```

```
'./near-wallet' ;
```

```
const
```

```
CONTRACT_ADDRESS
```

```
=
```

```
"nft.primitives.near" ; const wallet =
```

```
new
```

```
Wallet ( {
```

```
createAccessKeyFor :
```

```
CONTRACT_ADDRESS
```

```
} ) ;
```

```
await wallet . callMethod ( { method :
```

```
'nft_transfer' , args :
```

```
{ token_id :
```

```
"1" , receiver_id :
```

```
"bob.near" } , contractId :
```

```
CONTRACT_ADDRESS , deposit :
```

```
1 } ) ; TheWallet object comes from our quickstart template import
```

```
{
```

```
Wallet
```

```
}
```

```
from
```

```
'./near-wallet' ;
```

```

const
CONTRACT_ADDRESS

=

"x.paras.near" ; const wallet =

new

Wallet ( {

createAccessKeyFor :

CONTRACT_ADDRESS

} ) ;

await wallet . callMethod ( { method :

'nft_transfer' , args :

{ token_id :

"490641" , receiver_id :

"bob.near" } , contractId :

CONTRACT_ADDRESS , deposit :

1 } ) ; TheWallet object comes from ourquickstart template import

{

Wallet

}

from

'./near-wallet' ;

const

CONTRACT_ADDRESS

=

"thomasettorreiv.mintbase1.near" ; const wallet =

new

Wallet ( {

createAccessKeyFor :

CONTRACT_ADDRESS

} ) ;

await wallet . callMethod ( { method :

'nft_transfer' , args :

{ token_id :

"490641" , receiver_id :

"bob.near" } , contractId :

CONTRACT_ADDRESS , deposit :

1 } ) ; TheWallet object comes from ourquickstart template

```

[illegible]

1 ;

 $\{$ 

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

**[private]**

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

```
fn
```

```
nft_transfer_callback ( & self ,
```

## [callback\_result]

```
call_result :
```

```
Result < ( ) ,
```

```
PromiseError
```

```
)
```

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

```
{ log! ( "There was an error contacting NFT contract" ) ; } }
```

## Attaching NFTs to a Call

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

- CLI

near call nft\_transfer\_call '{"receiver\_id": "", "token\_id": "", "msg": ""}' --accountId--depositYocto 1 info Optionally, a [memo parameter](#) can be passed to provide more information to your contract.

## How Does it Work?

Assume you want to attach an NFT ( ) to a call on the receiver contract. The workflow is as follows:

1. You call `nft_transfer_call`
2. in the NFT-contract passing: the receiver, a message, and the token-id of .
3. The NFT contract transfers the NFT to the receiver.
4. The NFT contract calls `receiver.nft_on_transfer(sender, token-owner, token-id, msg)`
5. .
6. The NFT contract handles errors in the `nft_resolve_transfer`
7. `callback`.
8. The NFT contract returns `true`
9. if it succeeded.

## The `nft_on_transfer` method

From the workflow above it follows that the receiver we want to call needs to implement the `nft_on_transfer` method. When executed, such method will know:

- Who is sending the NFT, since it is a parameter
- Who is the current owner, since it is a parameter
- Which NFT was transferred, since it is a parameter.
- If there are any parameters encoded as a message

The `nft_on_transfer` must return `true` if the NFT has to be returned to the sender .

## Approving Users

You can authorize other users to transfer an NFT you own. This is useful, for example, to enable listing your NFT in a marketplace. In such scenario, you trust that the marketplace will only transfer the NFT upon receiving a certain amount of money in exchange.

- CLI

near call nft\_approve '{ "token\_id": "", "account\_id": "", "msg": "" }' --accountId--depositYocto 1 info If the `msg` parameter is included, then a cross-contract call will be made to `nft_on_approve(msg)` . Which in turn will make a `callback` `onft_resolve_transfer` in your NFT contract.

# List a NFT for sale

Basic NFT contracts following [the NEP-171 and NEP-177 standards](#) do not implement marketplace functionality.

For this purpose, there are ecosystem apps such as [Paras](#) or [Mintbase](#) , that use dedicated marketplace contracts.

In order to put a NFT for a sale on a marketplace you need to do two actions:

1. Cover data storage costs in the marketplace contract.
2. Approve the marketplace to sell the NFT in your NFT contract.
3. ❄ Component
4. WebApp
5. CLI
6. Paras
7. Mintbase

```
Near . call ( "marketplace.paras.near" , "storage_deposit" , { receiver_id :
```

```
"bob.near" } , undefined , 9390000000000000000 ) ;
```

```
Near . call ( "nft.primitives.near" , "nft_approve" , { token_id :
```

```
"1e95238d266e5497d735eb30" , account_id :
```

```
"marketplace.paras.near" , msg :
```

```
{ price :
```

```
"20000000000000000000000000000000" , market_type :
```

```
"sale" , ft_token_id :
```

```
"near" } } ) ; The method nft_approve will call nft_on_approve in marketplace.paras.near . Near . call ( "simple.market.mintbase1.near" , "deposit_storage" , { autotransfer :
```

```
true } , undefined , 9390000000000000000 ) ;
```

```
Near . call ( "nft.primitives.near" , "nft_approve" , { token_id :
```

```
"3c46b76cbd48e65f2fc88473" , account_id :
```

```
"simple.market.mintbase1.near" , msg :
```

```
{ price :
```

```
"20000000000000000000000000000000" } } ) ; The method nft_approve will call nft_on_approve in simple.market.mintbase1.near . * Paras * Mintbase
```

```
import
```

```
{
```

```
Wallet
```

```
}
```

```
from
```

```
'./near-wallet' ;
```

```
const
```

```
CONTRACT_ADDRESS
```

```
=
```

```
"marketplace.paras.near" ; const wallet =
```

```

new

Wallet ( {

createAccessKeyFor :

CONTRACT_ADDRESS

} ) ;

await wallet . callMethod ( { method :

'storage_deposit' , args :

{ receiver_id :

"bob.near" } , contractId :

CONTRACT_ADDRESS , gas :

3000000000000000 ,

// attached GAS (optional) deposit :

9390000000000000000

// attached deposit in yoctoNEAR (optional) } ) ;

await wallet . callMethod ( { method :

'nft_approve' , args :

{ token_id :

"1e95238d266e5497d735eb30" , account_id :

"marketplace.paras.near" , msg :

{ price :

"20000000000000000000000000000000" , market_type :

"sale" , ft_token_id :

"near" } } , contractId :

"nft.primitives.near" } ) ; TheWallet object comes from our quickstart template

Methodnft_approve of a NFT contract also calls thenft_on_approve method inmarketplace.paras.near as a callback. import

{

Wallet

}

from

'./near-wallet' ;

const

CONTRACT_ADDRESS

=

"simple.market.mintbase1.near" ; const wallet =

new

Wallet ( {

createAccessKeyFor :

```

```
CONTRACT_ADDRESS
```

```
}) ;
```

```
await wallet . callMethod ( { method :
```

```
'deposit_storage' , args :
```

```
{ autotransfer :
```

```
true } , contractId :
```

```
CONTRACT_ADDRESS , gas :
```

```
3000000000000000 ,
```

```
// attached GAS (optional) deposit :
```

```
9390000000000000000
```

```
// attached deposit in yoctoNEAR (optional) } ) ;
```

```
await wallet . callMethod ( { method :
```

```
'nft_approve' , args :
```

```
{ args :
```

```
{ token_id :
```

```
"3c46b76cbd48e65f2fc88473" , account_id :
```

```
"simple.market.mintbase1.near" , msg :
```

```
{ price :
```

```
"20000000000000000000000000000000" } } , contractId :
```

```
"nft.primitives.near" } ) ; TheWallet object comes from our quickstart template
```

Method `nft_approve` of a NFT contract also calls `thenft_on_approve` method in `simple.market.mintbase1.near` as a callback.

tip Check how to also do this using the [Mintbase JS](#) \* Paras \* Mintbase

```
near call marketplace.paras.near storage_deposit '{"receiver_id": "bob.near"}' --accountId bob.near --deposit 0.00939
```


```
near call nft.primitives.near nft_approve '{"token_id": "1e95238d266e5497d735eb30", "account_id":  
"marketplace.paras.near", "msg": {"price": "20000000000000000000000000000000", "market_type": "sale", "ft_token_id": "near"}}' --  
accountId bob.near Methodnft_approve of a NFT contract also calls thenft_on_approve method in marketplace.paras.near as  
a callback. near call simple.market.mintbase1.near deposit_storage '{"autotransfer": "true"}' --accountId bob.near --deposit  
0.00939
```

```
near call nft.primitives.near nft_approve '{"token_id": "3c46b76cbd48e65f2fc88473", "account_id":  
"simple.market.mintbase1.near", "msg": {"price": "20000000000000000000000000000000"}}' --accountId bob.near Methodnft_approve  
of a NFT contract also calls thenft_on_approve method in simple.market.mintbase1.near as a callback.
```

## Buy a NFT

Basic NFT contracts following [the NEP-171 and NEP-177 standards](#) do not implement marketplace functionality.

For this purpose, there are ecosystem apps such as [Paras](#) or [Mintbase](#) , that use dedicated marketplace contracts.

-  Component
- WebApp
- CLI
- Contract
- Paras
- Mintbase

```
const tokenData =
```



```

Near . call ( "x.paras.near" , "nft_buy" , { token_series_id :
"299102" , receiver_id :
"bob.near" , } , undefined , 2057400000000000000000000000000000
// NFT price + storage cost ) ; Example response:
"299102:1" const tokenData =
Near . call ( "simple.market.mintbase1.near" , "buy" , { nft_contract_id :
"rubennnnnnnnn.mintbase1.near" , token_id :
"38" , referrer_id :
null , } , undefined , 10000000000000000000000000000000000
// NFT price + storage cost (optional, depends on a contract) ) ; Example response:
{ "payout": { "rub3n.near": "88920000000000000000000000000000" , "rubenm4rcus.near": "85800000000000000000000000000000" } } * Paras *
Mintbase

import
{
  Wallet
}
from
'./near-wallet' ;

const
CONTRACT_ADDRESS
=
"x.paras.near" ; const wallet =
new
Wallet ( {
  createAccessKeyFor :
    CONTRACT_ADDRESS
} ) ;

await wallet . callMethod ( { method :
'nft_buy' , args :
{ token_series_id :
"299102" , receiver_id :
"bob.near" , } , contractId :
CONTRACT_ADDRESS , deposit :
2057400000000000000000000000000000
// attached deposit in yoctoNEAR, covers NFT price + storage cost } ) ; TheWallet object comes from our quickstart template

Example response "299102:1" import
{
  Wallet

```

[illegible]

**[near\_bindgen]**

```
[derive(BorshDeserialize, BorshSerialize)]
```

pub

```

struct
Contract
{ nft_marketplace_contract :
AccountId }
impl
Default
for
Contract
{ // The default trait with which to initialize the contract fn
default ( )
->
Self
{ Self
{ nft_marketplace_contract :
NFT_MARKETPLACE_CONTRACT . parse ( ) . unwrap ( ) } } }
// Validator interface, for cross-contract calls

```

## [ext\_contract(ext\_nft\_contract)]

```

trait
ExternalNftContract
{ fn
buy ( & self , nft_contract_id :
AccountId , token_id :
TokenId , ft_token_id :
Option < AccountId
, price :
Option < U128
)
->
Promise ; }
// Implement the contract structure

```

## [near\_bindgen]

```

impl
Contract
{

```

## [payable]

```

pub

fn

buy ( & mut

self , nft_contract_id :

AccountId , token_id :

TokenId , ft_token_id :

Option < AccountId

    , price :

Option < U128

    )

->

Promise

{ let promise =

ext_nft_contract :: ext ( self . nft_marketplace_contract . clone ( ) ) . with_static_gas ( Gas ( 30 * TGAS ) ) .
with_attached_deposit ( env :: attached_deposit ( ) ) . buy ( nft_contract_id , token_id , ft_token_id , price ) ;

return promise . then (

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

```

## [private]

```

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

fn

buy_callback ( & self ,

```

## [callback\_result]

```

call_result :

Result < ( ) ,

PromiseError

    )

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

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

```

## Additional Resources

1. [NFT Tutorial](#)
2. (NEAR examples, JavaScript SDK) - a set of tutorials that cover how to create a NFT contract using JavaScript.
3. [NFT Tutorial](#)
4. (NEAR examples, Rust SDK) - a set of tutorials that cover how to create a NFT contract using Rust.
5. [NFT Tutorial by Keypom](#)
6. (a fork of the NEAR example tutorial).
7. [Paras API documentation](#)
8. .
9. [Mintbase API documentation](#)
10. .
11. [Mintbase JS SDK](#)

12.
  - a set of methods to get data from blockchain, interact with Mintbase contracts, etc.
13. [Examples](#)
14. on how to mint NFT, query metadata, attach NFTs to a contract call using near-cli
15. . [Edit this page](#) Last updated on Feb 28, 2024 by Damian Parrino Was this page helpful? Yes No

[Previous Fungible Tokens \(FT\)](#) [Next Linkdrops](#)