

The Contract Class

Your contract's logic and state (storage) is defined by the [main class](#) , in which:

1. The attributes define the [contract's state](#)
2. The [initialization methods](#)
3. define how to initialize the contract's state
4. The public methods act as the contract's interface with the rest of the network

Defining the Contract

The contract is just another class, with its own attributes and methods. To differentiate it from other internal classes simply decorate it using the [NEAR Bindgen decorator/macro](#) .

- JavaScript
- Rust

contract-ts/src/contract.ts loading ... [See full example on GitHub](#) contract-rs/src/lib.rs loading ... [See full example on GitHub](#)
Under the hood, the NEAR Bindgen decorator/macro traverses the class, generating the necessary code to:

1. Transform the code into a valid NEAR contract.
2. Expose public methods, so they can be called externally.
3. Serialize objects for internal storage and communication with external actors.

The State

Each account has its own state (storage), which only they can modify but [anyone can see](#) .

The state is defined and modified through the main class' attributes .

Contracts [pay for their storage](#) by locking part of their balance. Currently it costs ~1 Ⓝ to store 100kb

Key-Value Storage The contract actually uses a key-value storage to persist values. This however is abstracted from you by the SDK through [serialization](#) . Prefer SDK Collections When defining attributes, always prefer [SDK collections](#) over native ones, since they are optimized for [serialization](#) .

Initializing the State

There are two ways to initialize the account's state, and they can co-exist:

1. An initialization method
2. that receives the attributes needed for the state
3. A default state
4. , which will be used until init
5. is invoked, or a method writes into the state

Initialization Method

To define an initialization method simply decorate it with the [initialization macro](#) .

The method will now be able to define the initial state's values, raising an error if invoked while the state is already initialized .

- JavaScript
- Rust

contract-ts/src/contract.ts loading ... [See full example on GitHub](#) info To make the initialization mandatory use `@NearBindgen({requireInit: true})` caution In JavaScript you must always define a [default state](#) contract-rs/src/lib.rs loading ... [See full example on GitHub](#) info To make the initialization mandatory use `#[derive(PanicOnDefault)]` in the contract's structure

Default State

Contracts can define a default state to use if no initialize method is called. This is, if any method is invoked before an init happens, the contract will use the default values.

Once any method writes into the state, the state will be considered initialized.

- JavaScript
- Rust

contract-ts/src/contract.ts loading ... [See full example on GitHub](#)
 defined by the initialization parameters in the class definition.

In JavaScript, the default state is

caution In Javascript you must always assign values to all the class' parameters . This ensures they get correctly [deserialized](#)
 to their intended type. contract-rs/src/lib.rs loading ... [See full example on GitHub](#)

Interface

All the public methods are exposed to the network as the contract's interface.

- JavaScript
- Rust

@ NearBindgen ({ }) class

Contract

{

@ initialize ({

...

}) init ({

...

})

{

/ public init method /

}

@ view ({ }) get_message ({

...

})

{

/ public view method /

}

@ call ({ }) add_message ({

...

})

{

/ public call method /

}

private

internal_search (

...

)

{

```

/ private internal method/
}

@ call ( { privateFunction :
true } ) set_owner ( {
...
} )
{
/ public, panics when caller is not the contract's account/
}}

```

[near_bindgen]

```

impl
Contract
{

```

[init]

```

pub
fn
init (
...
)
->
Self
{
/ public init method/
} pub
fn
get_message ( & self ,
...
)
{
/ public view method/
} pub
fn
add_message ( & mut
self ,
...

```

```

)
{
/ public call method /
} fn
internal_search ( & self ,
...
)
{
/ private internal method /
}

```

[private]

```

pub
fn
set_owner ( & mut
self ,
...
)
{
/ public, panics when caller is not the contract's account /
} }

```

Public Methods

Public methods can be categorized in three types: init methods, view methods, and call methods.

- Init Methods
 - : They define how to initialize the state of the contract.
- View Methods
 - : Donot mutate the state or call other contracts. They can be called for free by everyone, without needing a NEAR account.
- Call Methods
 - : They can mutate the state and perform [actions](#) such as calling other contracts.

caution By default view methods have 200TGas to execute, to increase this you can simply invoke them as call methods.
 danger By default init methods are public, make sure to [decorate them as private](#) , or [batch call the initialization on deploy](#)

Private Methods

Sometimes you will want some methods to remain public, but only be callable by the contract's account. Such is the case for example of [cross-contract callbacks](#) .

For this, you can use the `private` macro/decorator.

- JavaScript
- Rust

```
@ call ( { privateFunction :
true } ) callback (
...
) { // this method can only be called by the contract's account }
```

[private]

```
pub
fn
callback ( & mut
self ,
...
) { // this method can only be called by the contract's account }
```

Payable Methods

By default all methods panic if a user attaches money while calling them. To enable a method to receive money use the payable decorator.

- JavaScript
- Rust

```
@ call ( { payableFunction :
true } ) deposit_and_stake (
...
) { // this method can receive money from the user }
```

[payable]

```
pub
fn
deposit_and_stake ( & mut
self ,
...
) { // this method can receive money from the user }
```

Input & Return Types

The contract can receive and return any native type , including complex structures. However, since contracts communicate through their interface [using JSON](#) :

- Always prefer native types
- over SDK Collections
- in the input & return types.
- Replace u64
- /u128
- for strings
- (U64
- /U128
- in the Rust SDK). [Edit this page](#) Last updated on Feb 8, 2024 by matiasbenary Was this page helpful? Yes No

[Previous Modules, Types & Structs](#) [Next Environment](#)