

Private Methods

When using callbacks

Usually, when a contract has to have a callback for a remote cross-contract call, this callback method should only be called by the contract itself. It's to avoid someone else calling it and messing the state. Pretty common pattern is to have an assertion that validates that the direct caller (predecessor account ID) matches to the contract's account (current account ID). `Macro#[private]` simplifies it, by making it a single line macro instead and improves readability.

Use this annotation within the [near_bindgen macro](#) as follows:

[private]

```
pub
fn
my_method ( & mut
self )
{ ... } Which is equivalent to:
pub
fn
my_method ( & mut
self
)
{ if
near_sdk :: env :: current_account_id ( )
!=
near_sdk :: env :: predecessor_account_id ( )
{ near_sdk :: env :: panic_str ( "Method method is private" ) ; } ... } Now with this annotation, only the account of the contract
itself can call this method, either directly or through a promise.
```

Writing internal methods

Not all functions need to be exposed publicly. It may be beneficial to write private methods for helper or utility functions, for instance. There are three approaches to write internal methods:

1. Using `fn`
2. instead of `pub fn`

```
fn
helper_method ( a :
u8 , b :
u8 )
{ ... } 1. Using pub(crate) fn 2. . This may be helpful when an internal method is in a different module.
// Function that can be called in another Rust file
pub ( crate )
fn
get_first_name ( account :
Account )
```

{ ... } More information from the [official Rust docs](#) regarding public/private methods.

1. Separateimpl
2. block

Another way of not exporting methods is by having a separateimpl Contract section, that is not marked with#[near_bindgen].

[near_bindgen]

```
impl
```

```
Contract
```

```
{ pub
```

```
fn
```

```
increment ( & mut
```

```
self )
```

```
{ self . internal_increment ( ) ; } } impl
```

```
Contract
```

```
{ /// This methods is still not exported. pub
```

```
fn
```

```
internal_increment ( & mut
```

```
self )
```

```
{ self . counter +=
```

```
1 ; } } Edit this page Last updated on Aug 24, 2022 by Damián Parrino Was this page helpful? Yes No
```

[Previous Contract Mutability](#) [Next Payable Methods](#)