

Reducing a contract's size

Advice & examples

This page is made for developers familiar with lower-level concepts who wish to reduce their contract size significantly, perhaps at the expense of code readability.

Some common scenarios where this approach may be helpful:

- contracts intended to be tied to one's account management
- contracts deployed using a factory
- future advancements similar to the EVM on NEAR

There have been a few items that may add unwanted bytes to a contract's size when compiled. Some of these may be more easily swapped for other approaches while others require more internal knowledge about system calls.

Small wins

Using flags

When compiling a contract make sure to pass flag-C link-arg=-s to the rust compiler:

`RUSTFLAGS='-C link-arg=-s' cargo build --target wasm32-unknown-unknown --release` Here is the parameters we use for the most examples in `Cargo.toml` :

`[profile.release] codegen-units = 1 opt-level = "s" lto = true debug = false panic = "abort" overflow-checks = true` You may want to experiment with `using-opt-level = "z"` instead of `opt-level = "s"` to see if generates a smaller binary. See more details on this in [The Cargo Book Profiles section](#) . You may also reference this [Shrinking .wasm Size](#) resource.

Removing `rlib`

from the manifest

Ensure that your manifest (`Cargo.toml`) doesn't contain `rlib` unless it needs to. Some NEAR examples have included this:

Adds unnecessary bloat `[lib] crate-type = ["cdylib", "rlib"]` when it could be:

tip `[lib] crate-type = ["cdylib"]` 1. When using the Rust SDK, you may override the default JSON serialization to use [Borsh](#) 2. instead. [See this page](#) 3. for more information and an example. 4. When using assertions or guards, avoid using the standard `assert` 5. macros like `assert!` 6. `assert_eq!` 7. , or `assert_ne!` 8. as these may add bloat for information regarding the line number of the error. There are similar issues with `unwrap` 9. `expect` 10. , and Rust's `panic!()` 11. macro.

Example of a standard assertion:

Adds unnecessary bloat `assert_eq! (contract_owner , predecessor_account , "ERR_NOT_OWNER") ;` when it could be:

tip if `contract_owner != predecessor_account { env :: panic (b"ERR_NOT_OWNER") ; }` Example of removing `expect` :

Adds unnecessary bloat `let owner_id =`

`self . owner_by_id . get (& token_id) . expect ("Token not found") ;` when it could be:

tip fn

`expect_token_found < T`

`(option :`

`Option < T`

`)`

`->`

`T`

`{ option . unwrap_or_else (||`

```
env :: panic_str ( "Token not found" ) ) } let owner_id =
```

```
expect_token_found ( self . owner_by_id . get ( & token_id ) ) ; Example of changing standardpanic!() :
```

Adds unnecessary bloat panic! ("ERR_MSG_HERE") ; when it could be:

```
tip env :: panic_str ( "ERR_MSG_HERE" ) ;
```

Lower-level approach

For an `ino_std` approach to minimal contracts, observe the following examples:

- [Tiny contract](#)
- [NEAR ETH Gateway](#)
- [This YouTube video](#)
- where Eugene demonstrates a fungible token in `ino_std`
- mode. The code for this [example lives here](#)
- .
- [Examples using a project called nesdie](#)
- .
- Note that Aurora has found success using [rjson](#)
- as a lightweight JSON serialization crate. It has a smaller footprint than [serde](#)
- which is currently packaged with the Rust SDK. See [this example of rjson](#)
- in an Aurora repository, although implementation details will have to be gleaned by the reader and won't be expanded upon here. [This nesdie example](#)
- also uses the [miniserde crate](#)
- , which is another option to consider for folks who choose to avoid using the Rust SDK.

Information on system calls Expand to see what's available from `sys.rs` near `near-sdk/src/environment/sys.rs` loading ... [See full example on GitHub](#) [Edit this page](#) Last updated on Jan 19, 2024 by Damián Parrino Was this page helpful? Yes No

[Previous Best Practices](#) [Next About JS SDK](#)