한 번에 끝내는
블록체인 개발 A to Z

Chapter 4

How to Develop Solana Contract?

Chapter 4

How to Develop Solana Contract?

Develop With Rust 2

Depending on Rand

Programs are constrained to run deterministically, so random numbers are not available. Sometimes a program may depend on a crate that depends itself on rand even if the program does not use any of the random number functionality. If a program depends on rand, the compilation will fail because there is no get-random support for Solana.

```
getrandom = { version = "0.1.14", features = ["dummy"] }
```

```
getrandom = { version = "0.2.2", features = ["custom"] }
```

Logging

Rust's **println!** macro is computationally expensive and not supported. Instead the helper macro **msg!** is provided.

```
msg!("A string");
```

```
msg!(0_64, 1_64, 2_64, 3_64, 4_64);
```

```
msg!("Some variable: {:?}", variable);
```

Panicking

Rust's **panic!**, **assert!**, and internal panic results are printed to the program logs by default.

```
INFO  solana_runtime::message_processor] Finalized account CGLhHSuWsp1gT4B7MY2KACqp9RUwQRhcUFfVSuxpSa
INFO  solana_runtime::message_processor] Call BPF program CGLhHSuWsp1gT4B7MY2KACqp9RUwQRhcUFfVSuxpSaj
INFO  solana_runtime::message_processor] Program log: Panicked at: 'assertion failed: `(left == right
      left: `1`,
     right: `2`', rust/panic/src/lib.rs:22:5
INFO  solana_runtime::message_processor] BPF program consumed 5453 of 200000 units
INFO  solana_runtime::message_processor] BPF program CGLhHSuWsp1gT4B7MY2KACqp9RUwQRhcUFfVSuxpSajZ fai
```

Copy

Custom Panic Handler

Programs can override the default panic handler by providing their own implementation.

First define the custom-panic feature in the program's Cargo.toml

```
[features]
default = ["custom-panic"]
custom-panic = []
```

```
#[cfg(all(feature = "custom-panic", target_arch = "bpf"))]
#[no_mangle]
fn custom_panic(info: &core::panic::PanicInfo<'_>) {
    solana_program::msg!("program custom panic enabled");
    solana_program::msg!("{}", info);
}
```