echo2tutorial

What is the purpose of this document	When	Who
Contains step by step instructions for run and understand the “echo2” program.	September 21, 2023	huitemagico

Summary

1. What is the purpose of this document

This document is an extension of echo2 README.md and provides detailed explanations to help you understand the design of 'echo2'. It also explains how to install the software in your own environment,testing, and run the program.

2. What is “echo2”

echo2 is a "tiny piece of code for learning and practice soroban rust sdk".

3. Where is echo2 source and the elements explained in this document

You can find these elements at https://github.com/huitemagico/echo2

4. List of the elements

Name	**Function**
lib.rs	The ‘echo2’ program
test.rs	Test program
reset.sh	Shell for send message for reset
sorb.sh	Shell for build
compi.sh	Shell for compile
runseq.sh	Shell for send a sequence of messages

5. What is the function of the echo2 program

This program takes a text parameter as input from the caller, retrieves the previous message from persistent storage associated with the contract, and then saves the new message. So, with each subsequent call, 'echo2' returns 'the echo of your message' along with the previous message.

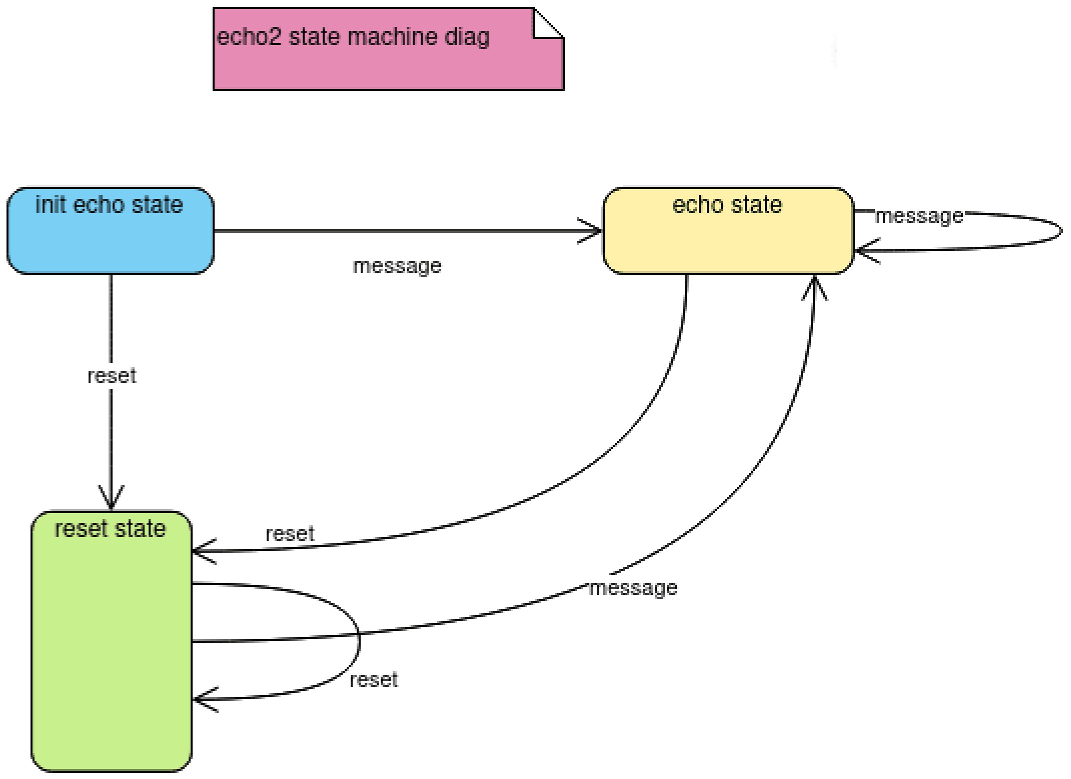
6. Echo2 as an finite state machine

Echo2 is a program that implements a basic state machine with two states: reset_state and echo_state. In the reset_state, the program does not process the previous message but instead resets it with a special meaning. If Echo2 receives another reset message while in the reset_state, it remains in that state. However, if it receives a message different from "reset," it transitions to the echo_state while saving the previous state and counter.

While in the echo_state, if Echo2 receives a general message that is different from "reset," it remains in that state. The machine continues to stay in the echo_state until it receives a "reset" message, at which point it transitions back to the reset_state.

The following diagram describes this.

Note: The explanation of a state machine is easiest using a "[finite state machine diagram]" for better understanding.



7. Steps:

In this tutorial, I'll show you how to run a Soroban contract using the Rust Soroban Library... from 'zero'.

By 'zero,' I mean: a. Installing Rust b. Installing Soroban tools c. Writing a Rust contract with the Soroban library d. Running the contract.

I will also provide some URLs with additional information for learning more about each of these subjects.

8. Tipographic conventions

Commands

I will use the command terminal, and to display the commands, I will use the following format. For instance, for 'print working directory' (pwd), it will appear like this:

```
ruser1@tortola:~$ pwd
```

The computer name is “tortola” and the user name is “ruser1”. (by the way, “tortola” is the name of a bird = zenaida articulata)

Output from the commands

For example, the output of pwd is:

```
/home/ruser1
```

If I need to highlight a certain part of the computer's output, I will mark it in yellow, like this:

```
/home/ruser1
```

9. Installing rust

Note: I am following the instructions at https://soroban.stellar.org/docs/getting-started/setup Note: The following instructions are for Ubuntu, but for another Unix-like OS, the process would be the same. For example, macOS Monterey 12.5.1.

10. The system I use is:

Ubuntu 23.04

11. Ubuntu user

I have created a user named 'ruser1'. The following commands are executed under the 'ruser1' account.

12. Installing Rust

Is it already installed, and where can I find it?

In some cases, perhaps before beginning this tutorial, you have already installed Rust.

a. Where is rust? easy way, call the rustc :

Case 1:

```
ruser1@tortola:~$ rustc -V
```

```
Command 'rustc' not found, but can be installed with:
snap install rustup  # version 1.24.3, or
apt  install rustc   # version 1.67.1+dfsg0ubuntu1-0ubuntu2
See 'snap info rustup' for additional versions.
```

In this case rust is not installed for the ruser1.

Case 2:

```
$ rustc -V
```

```
rustc 1.71.1 (eb26296b5 2023-08-03)
```

If you get a message like that, it means Rust is installed.

In the following, we assume that rust is not installed.

13. Installing rust.

For installing Rust:

In the same session of “ruser1”: open terminal and then copy the following:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Note: Please remember that I have followed the Soroban documentation, not the Ubuntu messages related to the installation of Rust.

You will see the following messages:

Note: I have omitted some lines from the output.

```
info: downloading installer

Welcome to Rust!

This will download and install the official compiler for the Rust
programming language, and its package manager, Cargo.

Rustup metadata and toolchains will be installed into the Rustup
home directory, located at:

  /home/ruser1/.rustup

...etc...

This path will then be added to your PATH environment variable by

modifying the profile files located at:

  /home/ruser1/.profile

...etc…

1) Proceed with installation (default)

2) Customize installation

3) Cancel installation

>1

info: profile set to 'default'

info: default host triple is x86_64-unknown-linux-gnu

...etc…

info: installing component 'rustc'

 ...

Rust is installed now. Great!
To get started you may need to restart your current shell.
This would reload your PATH environment variable to include
Cargo's bin directory ($HOME/.cargo/bin).
To configure your current shell, run:

source "$HOME/.cargo/env"
```

14. Inspecting the home directory:

```
ruser1@tortola:~$ pwd
```

```
/home/ruser1
```

```
ruser1@tortola:~$ ls -al
```

```
total 84
drwxr-x--- 18 ruser1 ruser1 4096 Sep 20 17:36  .
drwxr-xr-x  5 root   root   4096 Sep 20 17:07  ..
-rw-r--r--  1 ruser1 ruser1  220 Sep 20 17:07  .bash_logout
-rw-r--r--  1 ruser1 ruser1 3792 Sep 20 17:36  .bashrc
drwx------ 11 ruser1 ruser1 4096 Sep 20 17:34  .cache
drwxrwxr-x  3 ruser1 ruser1 4096 Sep 20 17:36  .cargo
drwx------ 11 ruser1 ruser1 4096 Sep 20 17:34  .config
drwx------  2 ruser1 ruser1 4096 Sep 20 17:08  .gnupg
drwx------  4 ruser1 ruser1 4096 Sep 20 17:08  .local
-rw-r--r--  1 ruser1 ruser1  828 Sep 20 17:36  .profile
drwxrwxr-x  6 ruser1 ruser1 4096 Sep 20 17:36  .rustup
```

15. Note: there are two new directorys.

Note: 'cargo' is the package manager for Rust. And now: Execute the command shown in the Rustup installation messages (refer to the highlighted message in the previous output):

```
ruser1@tortola:~$ source "$HOME/.cargo/env"
```

and now:

```
ruser1@tortola:~$ rustc -V
```

```
rustc 1.72.1 (d5c2e9c34 2023-09-13)
```

Good... but... where is Rustc installed?

```
ruser1@tortola:~$ which rustc
```

```
/home/ruser1/.cargo/bin/rustc
```

16. Installing the rust target

Following the doc at https://soroban.stellar.org/docs/getting-started/setup

```
ruser1@tortola:~$ rustup target add wasm32-unknown-unknown
```

```
info: downloading component 'rust-std' for 'wasm32-unknown-unknown'
info: installing component 'rust-std' for 'wasm32-unknown-unknown'
 16.8 MiB /  16.8 MiB (100 %)  16.2 MiB/s in  1s ETA:  0s
```

17. Install the Soroban CLI using cargo install.

```
ruser1@tortola:~$ cargo install --locked --version 0.9.4 soroban-cli
```

output a bunch of messages:

```
    Updating crates.io index
    Downloaded soroban-cli v0.9.4
    Downloaded 1 crate (77.2 KB) in 1.69s
    Installing soroban-cli v0.9.4

  ..etc...

  Finished release [optimized] target(s) in 1m 46s
  Installing /home/ruser1/.cargo/bin/soroban
  Installed package `soroban-cli v0.9.4` (executable `soroban`)
```

18. test the soroban cli with

```
ruser1@tortola:~$ soroban --version
```

showing :

```
soroban 0.9.4 (76cacc7cf885bd7d37756735fab332351d893406)
soroban-env 0.0.17 (400d806387140553e4e685d232deb3a807ec0e36)
soroban-env interface version 85899345971
stellar-xdr 0.0.17 (0f16673441898162c9996da6117be2280ef8fd84)
xdr next (e372df9f677961aac04c5a4cc80a3667f310b29f)
```

19. Do I need to install an IDE?

For simplicity, I will not install any IDE.

I will use vi editor (or any graphic editor) for this example.

20. Creating the rust library

Now, we create a new Rust library using the following:

```
ruser1@tortola:~$ cargo new --lib echo2
```

answer:

```
Created library `echo2` package
```

Note: here we are using the cargo command. See https://doc.rust-lang.org/cargo/appendix/glossary.html#cargo

The command above has created the 'echo2' directory. Upon inspecting this directory, we can see a bunch of files that the command has generated:

```
ruser1@tortola:~/echo2$ pwd
```

```
/home/ruser1/echo2
```

```
ruser1@tortola:~/echo2$ ls -alR |grep Cargo
```

```
-rw-rw-r--  1 ruser1 ruser1  174 Sep 20 17:59 Cargo.toml
```

```
ruser1@tortola:~/echo2$ ls -alR |grep src
```

```
drwxrwxr-x  2 ruser1 ruser1 4096 Sep 20 17:59 src
```

Note: There are a lot files (80+) created. From these I highlight the “Cargo.toml” file and the “src” directory.

21. Editing the package manager file manifest “Cargo.toml”

The cargo.toml file contains the instructions for cargo package manager.

We have to change the content of Cargo.toml with the following

```toml
[package]
name = "echo2"
version = "0.0.1"
authors = ["ruser1name"]
license = "Apache-2.0"
edition = "2021"
publish = false

[lib]

crate-type = ["cdylib"]
doctest = false


[dependencies]

soroban-sdk = { version = "0.9.2" }

[features]

testutils = ["soroban-sdk/testutils"]

[dev_dependencies]

soroban-sdk = { version = "0.9.2", features = ["testutils"] }

[profile.release]

opt-level = "z"
overflow-checks = true
debug = 0
strip = "symbols"
debug-assertions = false
panic = "abort"
codegen-units = 1
lto = true

[profile.release-with-logs]
inherits = "release"
debug-assertions = true
```

Note:please change authors = ["ruser1name"] with "your-name".
The other codes remains the same.

22. The program echo 2

The program must reside in the “src/lib.rs” file.

We will replace the original content of 'src/lib.rs' with the 'echo2' program. You can download the source code from GitHub at: https://github.com/huitemagico/echo2

or copy the following:

```
#![no_std]
// echo2-version for tutorial
use soroban_sdk::{contract,  contractimpl, symbol_short, vec, Env,
Symbol, Vec,String};
const COUNTER: Symbol = symbol_short!("COUNTER");
const OLD_MSG: Symbol = symbol_short!("OLD_MSG");
mod test;
#[contract]
pub struct Echo2Contract;
#[contractimpl]
impl Echo2Contract {
    pub fn echo2(env: Env, message: String) -> (u32,u32,Vec<String>)
{
     let _old_message = "nomessage";
     let resetmessage = String::from_slice(&env, "reset");
//save for work message received
     let  ln1:u32;
     ln1=message.len();
     let mut count: u32 =
env.storage().persistent().get(&COUNTER).unwrap_or(0); // If no
value set, assume 0.
     count += 1;
// Get the old message
     let mut old_message=env.storage()
                            .persistent()
                            .get(&OLD_MSG)
                            .unwrap_or(String::from_slice(&env,
"NoOldMessage0"));
     if message ==resetmessage {
               old_message=String::from_slice(&env,
"ResetMessageStored");
          }
     let msg = "echo2 v.1.1 27/08/2023";
     let sout = String::from_slice(&env, msg);
// Save new message in OLD_MSG storage
     env.storage().persistent().set(&OLD_MSG, &message);
     env.storage().persistent().set(&COUNTER, &count);
     return(   ln1, count,vec![&env, sout, old_message,message]    )

    }}
```

23. The “test” program.

To execute the 'contract,' we will use a caller. This function is provided by the 'test' program, which serves two purposes: a) To call the contract. b) To call the contract with specific parameters from which we can determine the contract's response.

The code for the 'test' program should be easy to follow and concise. For our purpose, we can use the following:

test.rs

```
#![cfg(test)]
use soroban_sdk::{String};
use super::*;

#[test]

fn test() {
    let env = Env::default();
    let contract_id = env.register_contract(None, Echo2Contract);
    let client = Echo2ContractClient::new(&env, &contract_id);
    // Step 1: Send first message
    let first_message=String::from_slice(&env, "reset");
   //expected response is [5,170,["echo2 v.1.1
27/08/2023","ResetMessageStored","reset"]]
   //comparing the first field with 5 for test evaluation ok
    let echo_response_tupla=client.echo2(&first_message);
    let expected_echo_response=5;
    assert_eq!(echo_response_tupla.0, expected_echo_response);

}
```

Note: the test.rs file must be in the src directory.

24. Compiling

To compile the contract, we will use the shell script 'compi.sh'.

```
ruser1@tortola:~/echo2$ cat compi.sh
```

```
cargo test -- --nocapture
```

We run the 'compi.sh' script, and it produces a bunch of output! That sounds good!

```
ruser1@tortola:~/echo2$ ./compi.sh
```

```
  Updating crates.io index
  Downloaded zeroize v1.6.0
  Downloaded spki v0.7.2
  Downloaded byteorder v1.4.3
  Downloaded serde_with_macros v3.3.0

...etc...

  Downloaded ecdsa v0.16.8
   Compiling proc-macro2 v1.0.67
   Compiling unicode-ident v1.0.12
   Compiling version_check v0.9.4
...etc...

   Compiling stellar-xdr v0.0.17
   Compiling soroban-env-common v0.0.17
   Compiling soroban-sdk-macros v0.9.2
…
   Compiling soroban-sdk v0.9.2

   Compiling echo2 v0.0.1 (/home/ruser1/echo2)

    Finished test [unoptimized + debuginfo] target(s) in 45.01s

     Running unittests src/lib.rs (target/debug/deps/echo2-
e1039e87b2f8645e)
```

So, we successfully test the contract!

25. Explanation of the “test”

The 'test.rs' caller executes the contract with a specific parameter, 'reset'. According to the contract's design, we know that with this parameter, the response from 'echo2' is a string of length 5. (The response is a tuple of objects, but for our purpose, we only consider the first element, which is the '0' element.)

Note: You can learn more about tuples in Rust at: https://doc.rust-lang.org/rust-by-example/primitives/tuples.html

26. Building

We will use the shell “sorb.sh”

```
ruser1@tortola:~/echo2$ cat sorb.sh
```

```
soroban contract build --profile release-with-logs
```

```
ruser1@tortola:~/echo2$ ./sorb.sh
```

```
cargo rustc --manifest-path=Cargo.toml --crate-type=cdylib --
target=wasm32-unknown-unknown –profile=release-with-logs
Downloaded wasm-bindgen v0.2.87
Downloaded wasm-bindgen-macro-support v0.2.87
Downloaded wasm-bindgen-backend v0.2.87

...etc…

Compiling echo2 v0.0.1 (/home/ruser1/echo2)
Finished release-with-logs [optimized] target(s) in 24.33s
```

27. Running the contract

Note: In the target directory, there must be the .wasm file generated in the former step. (The .wasm file is the WebAssembly binary generated by the Rust compiler from our Rust sources.) Now, we must run the contract and pass the parameters. The program 'echo2' expects two parameters. Note: The line below is in the program; for highlighting lines of the program, we will use the color as shown below.

```
pub fn echo2(env: Env, message: String)
```

The 'fn' indicates the main function. The 'pub' keyword makes the function accessible from an external module. The 'env' is the first parameter. The 'message' is the parameter we must provide.

So, to do that, we must call the contract as follows: In the 'message' parameter, we input a string, in this case, 'reset'.

```
soroban contract invoke \
    --wasm
target/wasm32-unknown-unknown/release-with-logs/echo2.wasm \
    --id 1 \
    -- \
    echo2\
    --message reset
```

Note: For easier operation, we have written a shell script ('reset.sh') with the instructions above. You can download it from the GitHub site or copy and paste it from the code above.

With these parameters, the program provides the following answer:

```
[5,4,["echo2 v.1.1 27/08/2023","ResetMessageStored","reset"]]
```

Note: To avoid the logging messages, we will redirect the output as follows.

```
ruser1@tortola:~/echo2$ ./reset.sh > pepe.txt
```

The output at the terminal is the following:

```
2023-09-20T21:41:55.413329Z  INFO soroban_cli::log::event:
log="[Diagnostic Event] topics:[fn_call,
Bytes(000000000000000000000000000000000000000000000000000000000000
01), echo2], data:\"reset\""
```

```
2023-09-20T21:41:55.413371Z  INFO soroban_cli::log::event:
log="[Diagnostic Event]
contract:0000000000000000000000000000000000000000000000000000000000000001, topics:[fn_return, echo2], data:[5, 2, [\"echo2 v.1.1
27/08/2023\", \"ResetMessageStored\", \"reset\"]]"

[5,2,["echo2 v.1.1 27/08/2023","ResetMessageStored","reset"]]
```

If we cat pepe.txt we see:

```
ruser1@tortola:~/echo2$ cat pepe.txt
```

```
[5,3,["echo2 v.1.1 27/08/2023","ResetMessageStored","reset"]]
```

28. Some details of the program

Please refer to the code in the preceding chapter or download it from GitHub.

About Storage.

```
let mut count: u32 =
env.storage().persistent().get(&COUNTER).unwrap_or(0);
```

This line retrieves the 'COUNTER' value from the contract's storage, which was previously stored during a prior execution."

For read about Storage:See https://docs.rs/soroban-sdk/latest/soroban_sdk/storage/index.html

In the program, I use 'persistent' storage because I need the program to 'remember' the previous message sent to it. (For experimentation or future tasks, I will explore and test other types of storage, although that is not covered in this document.)

https://soroban.stellar.org/docs/fundamentals-and-concepts/persisting-data

String

```
let msg = "echo2 v.1.1 27/08/2023";
let sout = String::from_slice(&env, msg);
```

Create string variable (constant) sout.

See
https://docs.rs/soroban-sdk/latest/soroban_sdk/struct.String.html#method.from_slice

Storage set value

```
env.storage().persistent().set(&OLD_MSG, &message);
```

See
https://docs.rs/soroban-sdk/latest/soroban_sdk/storage/struct.Storage.html

29. Calling several times the contract

I could call it by providing sequential parameters and observing the output. To make it easier, I have written another shell script: 'runseq.sh'

```
ruser1@tortola:~/echo2$ cat runseq.sh
```

```
soroban contract invoke \
    --wasm
target/wasm32-unknown-unknown/release-with-logs/echo2.wasm \
    --id 1 \
    -- \
    echo2\
    --message reset

soroban contract invoke \
    --wasm
target/wasm32-unknown-unknown/release-with-logs/echo2.wasm \
    --id 1 \
    -- \
    echo2\
    --message "On the last day of the world, I would want to plant
a tree."

soroban contract invoke \
    --wasm
target/wasm32-unknown-unknown/release-with-logs/echo2.wasm \
    --id 1 \
    -- \
    echo2\
    --message "Tell me what you see vanishing and I will tell you
who you are."

soroban contract invoke \
    --wasm
target/wasm32-unknown-unknown/release-with-logs/echo2.wasm \
    --id 1 \
    -- \
    echo2\
    --message "Poetry is a way of looking at the world for the 
first time."

soroban contract invoke \
    --wasm 
target/wasm32-unknown-unknown/release-with-logs/echo2.wasm \
    --id 1 \
    -- \
    echo2\
    --message "Now all my teachers are dead except silence."


soroban contract invoke \
    --wasm 
target/wasm32-unknown-unknown/release-with-logs/echo2.wasm \
    --id 1 \
    -- \
    echo2\
    --message "William S. Merwin"
```

The sequence of messages above, gives the following:

```
[5,16,["echo2 v.1.1 27/08/2023","ResetMessageStored","reset"]]
[59,17,["echo2 v.1.1 27/08/2023","reset","On the last day of the 
world, I would want to plant a tree."]]
[63,18,["echo2 v.1.1 27/08/2023","On the last day of the world, I 
would want to plant a tree.","Tell me what you see vanishing and I 
will tell you who you are."]]
[59,19,["echo2 v.1.1 27/08/2023","Tell me what you see vanishing and 
I will tell you who you are.","Poetry is a way of looking at the 
world for the first time."]]
[44,20,["echo2 v.1.1 27/08/2023","Poetry is a way of looking at the 
world for the first time.","Now all my teachers are dead except 
silence."]]
[17,21,["echo2 v.1.1 27/08/2023","Now all my teachers are dead 
except silence.","William S. Merwin"]]
```

30. Some urls

Topic	Url
storage	https://docs.rs/soroban-sdk/latest/soroban_sdk/storage/index.html
Persisting Data	https://soroban.stellar.org/docs/fundamentals-and-concepts/persisting-data
struct String	https://docs.rs/soroban-sdk/latest/soroban_sdk/struct.String.html
Soroban struct Vec	https://docs.rs/soroban-sdk/latest/soroban_sdk/struct.Vec.html
struct Env	https://docs.rs/soroban-sdk/latest/soroban_sdk/struct.Env.html
Soroban CLI	https://soroban.stellar.org/docs/getting-started/setup#install-the-soroban-cli
Soroban examples doc	https://soroban.stellar.org/docs/basic-tutorials/events
Soroban examples	https://github.com/stellar/soroban-examples
Soroban Rust sdk	https://soroban.stellar.org/docs/reference/sdks/rust

31. Recap

Throughout this tutorial, we have installed the necessary tools for writing a contract using the Soroban Rust SDK, and we have also learned how to test and run the program. This is just the beginning, but I hope it will make it easier for beginner readers to learn the Soroban Rust SDK.

Thank you for reading.