How to debug Stylus transactions using Cargo Stylus Replay

Debugging smart contracts can be challenging, especially when dealing with complex transactions. Thecargo-stylus crate simplifies the debugging process by allowing developers to replay Stylus transactions. This tool leverages GDB to provide an interactive debugging experience, enabling developers to set breakpoints, inspect state changes, and trace the execution flow step-by-step. This capability is crucial for identifying and resolving issues, ensuring that smart contracts function correctly and efficiently.

Overview

Cargo Stylus is a tool designed to simplify the development and debugging process for smart contracts written in Rust for the Stylus execution environment. One of its powerful features is thecargo stylus subcommand, which provides essential functionalities for developers:

1. Trace transactions
2. : Perform trace calls against Stylus transactions using Ethereum nodes'debug_traceTransaction
3. RPC. This feature enables developers to analyze the execution flow and state changes of their transactions in a detailed manner.
4. Debugging with GDB or LLDB
5. : Replay and debug the execution of a Stylus transaction using a debugger. This allows developers to set breakpoints, inspect variables, and step through the transaction execution line by line, providing an in-depth understanding of the transaction's behavior.

Replaying transactions

Requirements

- Rust
- (version 1.77 or higher)
- Crate
- :cargo-stylus
- GNU Debugger (GDB)
- (Linux) orLLDB
- (MacOS)
- Cast
- (an Ethereum CLI tool)
- Arbitrum RPC Provider
- with tracing endpoints enabled or alocal Stylus dev node

cargo stylus replay allows users to debug the execution of a Stylus transaction usingGDB orLLDB against the Rust source code.

Installation and setup

1. Install the required crates and debugger
2. : First, let's ensure that the following crates are installed:

cargo

install cargo-stylus If on Linux, install GDB if it's not already installed:

sudo

apt-get

install gdb If on MacOS, install LLDB if it's not already installed:

xcode-select --install 1. Deploy your Stylus contract 2. : For this guide, we demonstrate how to debug the execution of theincrement() 3. method in thestylus-hello-world 4. smart contract. In Rust, it looks something like this, withinsrc/lib.rs 5. :

[external]

impl Counter { .. . /// Increments number and updates its value in storage. pub fn increment (& mut self)

{ let number = self.number.get ( ) ; self.set_number (number + U256::from (1)) ; } .. . } Set your RPC endpoint to a node

withtracing enabled and your private key:

export

RPC_URL

.. . export

PRIV_KEY

.. . and deploy your contract:

cargo stylus deploy --private-key = PRIV_KEY

--endpoint

RPC_URL You should see an output similar to:

contract size: 4.0 KB wasm size: 12.1 KB contract size: 4.0 KB deployed code at address: 0x2c8d8a1229252b07e73b35774ad91c0b973ecf71 wasm already activated ! 1. Send a transaction 2. : First, set the address of the deployed contract as an environment variable:

export

ADDR

0x2c8d8a1229252b07e73b35774ad91c0b973ecf71 And send a transaction usingCast :

cast send --rpc-url

RPC_URL --private-key = PRIV_KEY

ADDR

"increment()" 1. Replay the transaction with the debugger 2. : Now, we can replay the transaction with cargo stylus and the debugger to inspect each step of it against our source code. Make sure GDB is installed and that you are on a Linux, x86 system. 3. Also, you should set the transaction hash as an environment variable:

export

TX_HASH

0x18b241841fa0a59e02d3c6d693750ff0080ad792204aac7e5d4ce9e20c466835 And replay the transaction:

cargo stylus replay --tx = TX_HASH

--endpoint

RPC_URL --use-native-tracer Options:

--tx: Specifies the transaction hash to replay. --endpoint: Specifies the RPC endpoint for fetching transaction data. --use-native-tracer: Uses the native Stylus tracer instead of the default JS tracer. The native tracer has broader support from RPC providers. Note: The--use-native-tracer flag usesstylusTracer instead ofjsTracer , which is required for tracing Stylus transactions on most RPC providers. See more detailsbelow .

The debugger will load and set a breakpoint automatically at theuser_entrypoint internal Stylus function. While the examples below showcase GDB commands, you can find the LLDB equivalentshere .

[Detaching after vfork from child process 370003]

Thread 1

"cargo-stylus" hit Breakpoint 1 , stylus_hello_world::user_entrypoint (len = 4) at src/lib.rs:38 38

[entrypoint]

(gdb) 1. Debugging 2. : Now, set a breakpoint at theincrement() 3. method:

(gdb) b stylus_hello_world::Counter::increment Breakpoint 2 at 0x7ffff7e4ee33: file src/lib.rs, line 69 . Then, typec to continue the execution and you will reach that line whereincrement is called:

(gdb) c Once you reach theincrement method, inspect the state:

Thread 1

"cargo-stylus" hit Breakpoint 2 , stylus_hello_world::Counter::increment (self = 0x7fffffff9ae8) at src/lib.rs:69 69

let number = self.number.get ( ) ; (gdb) p number

Trace a transaction

For traditional tracing,cargo stylus supports calls todebug_traceTransaction . To trace a transaction, you can use the following command:

cargo stylus trace [OPTIONS]

--tx

< TX

    --use-native-tracer Options:

-e, --endpoint

< ENDPOINT

    RPC endpoint [default: http://localhost:8547] -t, --tx

< TX

    Tx to replay -p, --project

< PROJECT

    Project path [default: .] -h, --help Print help -V, --version Print version --use-native-tracer Uses the native Stylus tracer instead of the default JS tracer. The native tracer has broader support from RPC providers. Run the following command to obtain a trace output:

cargo stylus trace --tx = TX_HASH

--endpoint

RPC_URL --use-native-tracer This will produce a trace of the functions called andink left along each method:

[{ "args" : [0,0 ,0,4] , "endInk" :846200000, "name" : "user_entrypoint" , "outs" : [ ] , "startInk" :846200000 } , { "args" : [ ] , "endInk" :846167558, "name" : "msg_reentrant" , "outs" : [0,0 ,0,0] , "startInk" :846175958 } , { "args" : [ ] , "endInk" :846047922, "name" : "read_args" , "outs" : [208,157 ,224,138] , "startInk" :846061362 } , { "args" : [ ] , "endInk" :845914924, "name" : "msg_value" , "outs" : [0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] , "startInk" :845928364 } , { "args" : [0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] , "endInk" :227196069, "name" : "storage_load_bytes32" , "outs" : [0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] , "startInk" :844944549 } , { "args" : [0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1] , "endInk" :226716083, "name" : "storage_cache_bytes32" , "outs" : [ ] , "startInk" :226734563 } , { "args" : [0] , "endInk" :226418732, "name" : "storage_flush_cache" , "outs" : [ ] , "startInk" :226486805 } , { "args" : [ ] , "endInk" :226362319, "name" : "write_result" , "outs" : [ ] , "startInk" :226403481 } , { "args" : [ ] , "endInk" :846200000, "name" : "user_returned" , "outs" : [0,0 ,0,0] , "startInk" :846200000 }]

RPC endpoint compatibility

Bothcargo stylus trace andcargo stylus replay require an RPC endpoint that supportsdebug_traceTransaction . By default, thejsTracer type is used, which is not supported by most RPC providers. If the--use-native-tracer flag is used,

thestylusTracer type is used, which is supported by many RPC providers. BothjsTracer andstylusTracer are available on local nodes, butstylusTracer is more efficient. See thislist of RPC providers for tracing support. Edit this page Last updatedonJan 27, 2025 Previous Overview Next Verify contracts