ABI Encode

TheABI Encode has 2 types which areencode andencode_packed .

- encode
- will concatenate all values and add padding to fit into 32 bytes for each values.
- encode_packed
- will concatenate all values in the exact byte representations without padding. (For example,encode_packed("a", "bc") == encode_packed("ab", "c")
-)

Suppose we have a tuple of values:(target, value, func, data, timestamp) to encode, and theiralloy primitives type are(Address, U256, String, Bytes, U256) .

Firstly we need to import those types we need fromalloy_primitives ,stylus_sdk::abi andalloc::string :

note This code has yet to be audited. Please use at your own risk. // Import items from the SDK. The prelude contains common traits and macros. use

stylus_sdk :: { alloy_primitives :: { U256 ,

Address ,

FixedBytes } ,

abi :: Bytes ,

prelude :: * } ; // Import String from alloc use

alloc :: string :: String ; Secondly because we will use the methodabi_encode_sequence andabi_encode_packed underalloy_sol_types to encode data, we also need to import the types fromalloy_sol_types :

// Becauce the naming of alloy_primitives and alloy_sol_types is the same, so we need to re-name the types in alloy_sol_types use

alloy_sol_types :: { sol_data :: { Address

as

SOLAddress ,

String

as

SOLString ,

Bytes

as

SOLBytes ,

* } ,

SolType } ;

encode

Thenencode them:

// define sol types tuple type

TxIdHashType

=

(SOLAddress ,

Uint < 256

,

SOLString ,

SOLBytes ,

Uint < 256

) ; // set the tuple let tx_hash_data =

(target , value , func , data , timestamp) ; // encode the tuple let tx_hash_bytes =

TxIdHashType :: abi_encode_sequence (& tx_hash_data) ;

encode_packed

There are 2 methods toencode_packed data:

1. encode_packed
2. them:

// define sol types tuple type

TxIdHashType

=

(SOLAddress ,

Uint < 256

,

SOLString ,

SOLBytes ,

Uint < 256

) ; // set the tuple let tx_hash_data =

(target , value , func , data , timestamp) ; // encode the tuple let tx_hash_data_encode_packed =

TxIdHashType :: abi_encode_packed (& tx_hash_data) ; 1. We can also use the following method toencode_packed 2. them:

let tx_hash_data_encode_packed =

[& target . to_vec ( ) ,

& value . to_be_bytes_vec ( ) , func . as_bytes ( ) ,

& data . to_vec ( ) ,

& timestamp . to_be_bytes_vec ( )] . concat ( ) ;

Full Example code:

src/main.rs

// Allow cargo stylus export-abi to generate a main function.

![cfg_attr(not(feature =

"export-abi"), no_main)] extern

crate

alloc ;

/// Import items from the SDK. The prelude contains common traits and macros. use

stylus_sdk :: { alloy_primitives :: { U256 ,

Address ,

FixedBytes } ,

abi :: Bytes ,

prelude :: * } ; use

alloc :: string :: String ; // Becauce the naming of alloy_primitives and alloy_sol_types is the same, so we need to re-name the types in alloy_sol_types use

alloy_sol_types :: { sol_data :: { Address

as

SOLAddress ,

String

as

SOLString ,

Bytes

as

SOLBytes ,

* } ,

SolType } ; use

sha3 :: { Digest ,

Keccak256 } ;

// Define some persistent storage using the Solidity ABI. //Encoder will be the entrypoint.

[storage]

[entrypoint]

pub

struct

Encoder ;

impl

Encoder

{ fn

keccak256 (& self , data :

Bytes)

->

FixedBytes < 32

{ // prepare hasher let

mut hasher =

Keccak256 :: new ( ) ; // populate the data hasher . update (data) ; // hashing with keccack256 let result = hasher . finalize ( ) ; // convert the result hash to FixedBytes<32> let result_vec = result . to_vec ( ) ; FixedBytes :: < 32

:: from_slice (& result_vec) } }

/// Declare that Encoder is a contract with the following external methods.

[public]

impl

Encoder

{

// Encode the data and hash it pub

fn

encode (& self , target :

Address , value :

U256 , func :

String , data :

Bytes , timestamp :

U256)

->

Vec < u8

{ // define sol types tuple type

TxIdHashType

=

(SOLAddress ,

Uint < 256

,

SOLString ,

SOLBytes ,

Uint < 256

) ; // set the tuple let tx_hash_data =

(target , value , func , data , timestamp) ; // encode the tuple let tx_hash_data_encode =

TxIdHashType :: abi_encode_params (& tx_hash_data) ; tx_hash_data_encode }

// Packed encode the data and hash it, the same result with the following one pub

fn

packed_encode (& self , target :

Address , value :

U256 , func :

String , data :

Bytes , timestamp :

U256) ->

Vec < u8

{ // define sol types tuple type

TxIdHashType

=

(SOLAddress ,

Uint < 256

,

SOLString ,

SOLBytes ,

Uint < 256

) ; // set the tuple let tx_hash_data =

(target , value , func , data , timestamp) ; // encode the tuple let tx_hash_data_encode_packed =

TxIdHashType :: abi_encode_packed (& tx_hash_data) ; tx_hash_data_encode_packed }

// Packed encode the data and hash it, the same result with the above one pub

fn

packed_encode_2 (& self , target :

Address , value :

U256 , func :

String , data :

Bytes , timestamp :

U256) ->

Vec < u8

{ // set the data to arrary and concat it directly let tx_hash_data_encode_packed =

[& target . to_vec ( ) ,

& value . to_be_bytes_vec ( ) , func . as_bytes ( ) ,

& data . to_vec ( ) ,

& timestamp . to_be_bytes_vec ( )] . concat ( ) ; tx_hash_data_encode_packed }

// The func example: "transfer(address,uint256)" pub

fn

encode_with_signature (& self , func :

String , address :

Address , amount :

U256)

->

Vec < u8

{ type

TransferType

=

(SOLAddress ,

Uint < 256

) ; let tx_data =

(address , amount) ; let data =

TransferType :: abi_encode_params (& tx_data) ; // Get function selector let hashed_function_selector =

self . keccak256 (func . as_bytes ( ) . to_vec ( ) . into ( )) ; // Combine function selector and input data (use abi_packed way) let calldata =

[& hashed_function_selector [.. 4] ,

& data] . concat ( ) ; calldata }

}

Cargo.toml

[package] name

=

"stylus-encode-hashing" version

=

"0.1.7" edition

=

"2021" license

=

"MIT OR Apache-2.0" keywords

=

["arbitrum" ,

"ethereum" ,

"stylus" ,

"alloy"]

[dependencies] alloy-primitives

=

"=0.7.6" alloy-sol-types

=

"=0.7.6" mini-alloc

=

"0.4.2" stylus-sdk

=

"0.6.0" hex

=

"0.4.3" sha3

=

"0.10"

[dev-dependencies] tokio

=

{

version

=

"1.12.0" ,

features

=

["full"]

} ethers

=

"2.0" eyre

=

"0.6.8"

[features] export-abi

=

["stylus-sdk/export-abi"]

[lib] crate-type

=

["lib" ,

"cdylib"]

[profile.release] codegen-units

=

1 strip

=

true lto

=

true panic

=

"abort" opt-level

=

"s" Edit this page Previous Function Selector Next Abi Decode