ABI Decode

Thedecode can not be used forencode_packed data because it ignores padding when encode. (For more information you can refer toABI Encode)

So here we show an example for usingdecode on data encoded withabi_encode_sequence :

```
note This code has yet to be audited. Please use at your own risk. // This should always return true pub

fn

encode_and_decode ( & self , target :

Address , value :

U256 , func :

String , data :

Bytes , timestamp :

U256 )

->

Result < bool ,

HasherError

{ // define sol types tuple type

TxIdHashType

=

( SOLAddress ,

Uint < 256

   ,

SOLString ,

SOLBytes ,

Uint < 256

   ) ; // because the abi_encode_sequence will return alloy_primitives::Bytes rather than stylus_sdk::bytes, so we
   need to make sure the input and return types are the same let primative_data =

alloy_primitives :: Bytes :: copy_from_slice ( & data ) ; // set the tuple let tx_hash_data =

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

TxIdHashType :: abi_encode_sequence ( & tx_hash_data ) ;

let validate =

true ;

// Check the result match

TxIdHashType :: abi_decode_sequence ( & tx_hash_data_encode , validate )

{ Ok ( res )

=>

Ok ( res == tx_hash_data ) , Err ( _ )

=>

{ return
```

Err (HasherError :: DecodedFailed (DecodedFailed { })) ; } , } }

Full Example code:

src/lib.rs

![cfg_attr(not(any(feature =

"export-abi" , test)), no_main)] extern

crate

alloc ;

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

stylus_sdk :: { alloy_primitives :: { U256 ,

Address } ,

prelude :: * } ; // Because the naming of alloy_primitives and alloy_sol_types is the same, we need to rename the types in alloy_sol_types. use

alloy_sol_types :: { sol_data :: { Address

as

SOLAddress ,

* } ,

SolType , sol } ;

// Define error sol!

{ error DecodedFailed ( ) ; }

// Error types for the MultiSig contract

[derive(SolidityError)]

pub

enum

DecoderError { DecodedFailed (DecodedFailed) }

[storage]

[entrypoint]

pub

struct

Decoder ;

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

[public]

impl

Decoder

```
{ // This should always return true pub
fn
encode_and_decode ( & self , address :
Address , amount :
U256 )
->
Result < bool ,
DecoderError
{ // define sol types tuple type
TxIdHashType
=
( SOLAddress ,
Uint < 256
        ) ; // set the tuple let tx_hash_data =
( address , amount ) ; // encode the tuple let tx_hash_data_encode =
TxIdHashType :: abi_encode_params ( & tx_hash_data ) ;
let validate =
true ;
// Check the result match
TxIdHashType :: abi_decode_params ( & tx_hash_data_encode , validate )
{ Ok ( res )
=>
Ok ( res == tx_hash_data ) , Err ( _ )
=>
{ return
Err ( DecoderError :: DecodedFailed ( DecodedFailed { } ) ) ; } , } }
}
```

Cargo.toml

```
[ package ] name =
"stylus-decode-hashing" version =
"0.1.0" edition =
"2021"
[ dependencies ] alloy - primitives =
"=0.7.6" alloy - sol - types =
"=0.7.6" mini - alloc =
"0.4.2" stylus - sdk =
"0.5.1"
```

[features] export - abi =

["stylus-sdk/export-abi"] debug =

["stylus-sdk/debug"]

[lib] crate - type

=

["lib" ,

"cdylib"]

[profile . release] codegen - units =

1 strip =

true lto =

true panic =

"abort" opt - level =

"s" Edit this page Previous Abi Encode Next Hashing