Primitive Data Types

TheStylus SDK makes use of the popularAlloy library (from the developers ofethers-rs andFoundry) to represent various native Solidity types as Rust types and to seamlessly convert between them when needed. These are needed since there are a number of custom types (like address) and large integers that are not natively supported in Rust.

In this section, we'll focus on the following types:

- U256
- I256
- Address
- Boolean
- Bytes

More in-depth documentation about the available methods and types in the Alloy library can be found in their docs. It also helps to cross-reference with Solidity docs if you don't already have a solid understanding of those types.

Learn More

- Alloy docs (v0.7.6)
-
    - Address
-
    - Signed
-
    - Uint
- Stylus Rust SDK
-
    - Bytes
- Solidity docs (v0.8.19)

Integers

Alloy defines a set of convenient Rust types to represent the typically sized integers used in Solidity. The typeU256 represents a 256-bitunsigned integer, meaning it cannot be negative. The range for aU256 number is 0 to 2^256 - 1.

Negative numbers are allowed for I types, such asI256 . These represent signed integers.

- U256
- maps touint256
- ...I256
- maps toint256
- U128
- maps touint128
- ...I128
- maps toint128
- ...
- U8
- maps touint8
- ...I8
- maps toint8

Integer Usage

note This code has yet to be audited. Please use at your own risk. // Unsigned let eight_bit :

U8

=

U8 :: from (1) ; let two_fifty_six_bit :

U256

=

U256 :: from (0xff_u64) ;

// Out: Stylus says: '8-bit: 1 | 256-bit: 255' console! ("8-bit: {} | 256-bit: {}" , eight_bit , two_fifty_six_bit) ;

// Signed let eight_bit :

I8

=

I8 :: unchecked_from (- 1) ; let two_fifty_six_bit :

I256

=

I256 :: unchecked_from (0xff_u64) ;

// Out: Stylus says: '8-bit: -1 | 256-bit: 255' console! ("8-bit: {} | 256-bit: {}" , eight_bit , two_fifty_six_bit) ;

Expanded Integer Usage

// Use try_from if you're not sure it'll fit let a =

I256 :: try_from (20003000) . unwrap ( ) ; // Or parse from a string let b =

"100" . parse :: < I256

( ) . unwrap ( ) ; // With hex characters let c =

"-0x138f" . parse :: < I256

( ) . unwrap ( ) ; // Underscores are ignored let d =

"1_000_000" . parse :: < I256

( ) . unwrap ( ) ;

// Math works great let e = a * b + c - d ; // Out: Stylus says: '20003000 * 100 + -5007 - 1000000 = 1999294993' console! ("{} * {} + {} - {} = {}" , a , b , c , d , e) ;

// Useful constants let f =

I256 :: MAX ; let g =

I256 :: MIN ; let h =

I256 :: ZERO ; let i =

I256 :: MINUS_ONE ;

// Stylus says: '5789...9967, -5789...9968, 0, -1' console! ("{f}, {g}, {h}, {i}") ; // As hex: Stylus says: '0x7fff...ffff, 0x8000...0000, 0x0, 0xffff...ffff' console! ("{:#x}, {:#x}, {:#x}, {:#x}" , f , g , h , i) ;

Address

Ethereum addresses are 20 bytes in length, or 160 bits. Alloy provides a number of helper utilities for converting to addresses from strings, bytes, numbers, and addresses.

Address Usage

// From a 20 byte slice, all 1s let addr1 =

Address :: from ([0x11 ;

20]) ; // Out: Stylus says: '0x1111111111111111111111111111111111111111' console! ("{addr1}") ;

// Use the address! macro to parse a string as a checksummed address let addr2 =

address! ("d8da6bf26964af9d7eed9e03e53415d37aa96045") ; // Out: Stylus says: '0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045' console! ("{addr2}") ;

// Format compressed addresses for output // Out: Stylus says: '0xd8dA…6045' console! ("{addr2:#}") ;

Boolean

Use native Rust primitives where it makes sense and where no equivalent Alloy primitive exists.

Boolean Usage

let frightened :

bool

=

true ; // Out: Stylus says: 'Boo! Did I scare you?' console! ("Boo! Did I scare you?") ;

let response =

match frightened { true

=>

"Yes!" . to_string ( ) , false

=>

"No!" . to_string ( ) , } ;

// Out: Stylus says: 'Yes!' console! ("{response}") ;

Bytes

The Stylus SDK provides this wrapper type aroundVecto represent abytes value in Solidity.

let vec =

vec! [108 ,

27 ,

56 ,

87] ; let b =

Bytes :: from (vec) ; // Out: Stylus says: '0x6c1b3857' console! (String :: from_utf8_lossy (b . as_slice ( ))) ;

let b =

Bytes :: from (b"Hello!" . to_vec ( )) ; // Out: Stylus says: 'Hello!' console! (String :: from_utf8_lossy (b . as_slice ( ))) ;
Note: Return theBytes type on your Rust function if you want to return the ABIbytes memory type.

Boilerplate

src/lib.rs

![cfg_attr(not(any(feature =

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

crate

alloc ; use

alloc :: { string :: ToString ,

vec :: Vec } ;

use

stylus_sdk :: { alloy_primitives :: { address ,

Address ,

I256 ,

I8 ,

U256 ,

U8 } , console ,

prelude :: * ,

ArbResult } ;

[storage]

[entrypoint]

pub

struct

Data

{

}

[public]

impl

Data

{ fn

user_main (_input :

Vec < u8

)

->

ArbResult

{ // Use native Rust primitives where they make sense // and where no equivalent Alloy primitive exists let frightened :

bool

=

true ; // Out: Stylus says: 'Boo! Did I scare you?' console! ("Boo! Did I scare you?") ;

let _response =

match frightened { true

=>

"Yes!" . to_string ( ) , false

=>

"No!" . to_string ( ) , } ;

// Out: Stylus says: 'Yes!' console! ("{_response}") ;

// U256 stands for a 256-bit *unsigned* integer, meaning it cannot be // negative. The range for a U256 number is 0 to 2^256 -

1. Alloy provides // a set of unsigned integer types to represent the various sizes available // in the EVM. // U256 maps to uint256 // U128 maps to uint128 // ... // U8 maps to uint8 let _eight_bit :

U8

=

U8 :: from (1) ; let _two_fifty_six_bit :

U256

=

U256 :: from (0xff_u64) ;

// Out: Stylus says: '8-bit: 1 | 256-bit: 255' console! ("8-bit: {} | 256-bit: {}" , _eight_bit , _two_fifty_six_bit) ;

// Negative numbers are allowed for I types. These represent signed integers. // I256 maps to int256 // I128 maps to int128 // ... // I8 maps to int8 let _eight_bit :

I8

=

I8 :: unchecked_from (- 1) ; let _two_fifty_six_bit :

I256

=

I256 :: unchecked_from (0xff_u64) ;

// Out: Stylus says: '8-bit: -1 | 256-bit: 255' console! ("8-bit: {} | 256-bit: {}" , _eight_bit , _two_fifty_six_bit) ;

// Additional usage of integers

// Use try_from if you're not sure it'll fit let a =

I256 :: try_from (20003000) . unwrap ( ) ; // Or parse from a string let b =

"100" . parse :: < I256

( ) . unwrap ( ) ; // With hex characters let c =

"-0x138f" . parse :: < I256

( ) . unwrap ( ) ; // Underscores are ignored let d =

"1_000_000" . parse :: < I256

( ) . unwrap ( ) ;

// Math works great let _e = a * b + c - d ; // Out: Stylus says: '20003000 * 100 + -5007 - 1000000 = 1999294993' console! ("{} * {} + {} - {} = {}" , a , b , c , d , _e) ;

// Useful constants let _f =

I256 :: MAX ; let _g =

I256 :: MIN ; let _h =

I256 :: ZERO ; let _i =

I256 :: MINUS_ONE ;

// Stylus says: '5789...9967, -5789...9968, 0, -1' console! ("{_f}, {_g}, {_h}, {_i}") ; // As hex: Stylus says: '0x7fff...ffff, 0x8000...0000, 0x0, 0xffff...ffff' console! ("{:#x}, {:#x}, {:#x}, {:#x}" , _f , _g , _h , _i) ;

// Ethereum addresses are 20 bytes in length, or 160 bits. Alloy provides a number of helper utilities for converting to addresses from strings, bytes, numbers, and addresses

// From a 20 byte slice, all 1s let _addr1 =

Address :: from ([0x11 ;

20]) ; // Out: Stylus says: '0x1111111111111111111111111111111111111111' console! ("{_addr1}") ;

// Use the address! macro to parse a string as a checksummed address let _addr2 =

address! ("d8da6bf26964af9d7eed9e03e53415d37aa96045") ; // Out: Stylus says: '0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045' console! ("{_addr2}") ;

// Format compressed addresses for output // Out: Stylus says: '0xd8dA…6045' console! ("{_addr2:#}") ;

Ok (Vec :: new ( )) } }

Cargo.toml

[package] name

=

"stylus_data_example" 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"

[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 Hello World Next Variables