ERC-20

Any contract that follows theERC-20 standard is an ERC-20 token.

ERC-20 tokens provide functionalities to

- transfer tokens
- allow others to transfer tokens on behalf of the token holder

Here is the interface for ERC-20.

interface

IERC20

{ function

totalSupply ( )

external

view

returns

(uint256) ; function

balanceOf (address account)

external

view

returns

(uint256) ; function

transfer (address recipient ,

uint256 amount) external returns

(bool) ; function

allowance (address owner ,

address spender) external view returns

(uint256) ; function

approve (address spender ,

uint256 amount)

external

returns

(bool) ; function

transferFrom (address sender ,

address recipient ,

uint256 amount) external returns

(bool) ; } Example implementation of an ERC-20 token contract written in Rust.

src/erc20.rs

note This code has yet to be audited. Please use at your own risk. //! Implementation of the ERC-20 standard //! //! The eponymous [Erc20] type provides all the standard methods, //! and is intended to be inherited by other contract types. //! //! You can configure the behavior of [Erc20] via the [Erc20Params] trait, //! which allows specifying the name, symbol, and

decimals of the token. //! //! Note that this code is unaudited and not fit for production use.

// Imported packages use

alloc :: string :: String ; use

alloy_primitives :: { Address ,

U256 } ; use

alloy_sol_types :: sol ; use

core :: marker :: PhantomData ; use

stylus_sdk :: { evm , msg , prelude :: * , } ;

pub

trait

Erc20Params

{ /// Immutable token name const

NAME :

& 'static

str ;

/// Immutable token symbol const

SYMBOL :

& 'static

str ;

/// Immutable token decimals const

DECIMALS :

u8 ; }

sol_storage!

{ /// Erc20 implements all ERC-20 methods. pub

struct

Erc20 < T

{ /// Maps users to balances mapping (address => uint256) balances ; /// Maps users to a mapping of each spender's allowance mapping (address =>

mapping (address => uint256)) allowances ; /// The total supply of the token uint256 total_supply ; /// Used to allow [Erc20Params] PhantomData < T

phantom ; } }

// Declare events and Solidity error types sol!

{ event Transfer (address indexed from , address indexed to , uint256 value) ; event Approval (address indexed owner , address indexed spender , uint256 value) ;

error InsufficientBalance (address from , uint256 have , uint256 want) ; error InsufficientAllowance (address owner , address spender , uint256 have , uint256 want) ; }

/// Represents the ways methods may fail.

[derive(SolidityError)]

pub

enum

Erc20Error

{ InsufficientBalance (InsufficientBalance) , InsufficientAllowance (InsufficientAllowance) , }

// These methods aren't exposed to other contracts // Methods marked as "pub" here are usable outside of the erc20 module (i.e. they're callable from lib.rs) // Note: modifying storage will become much prettier soon impl < T :

Erc20Params

Erc20 < T

{ /// Movement of funds between 2 accounts /// (invoked by the external transfer() and transfer_from() functions) pub

fn

_transfer (& mut

self , from :

Address , to :

Address , value :

U256 ,)

->

Result < ( ) ,

Erc20Error

{ // Decreasing sender balance let

mut sender_balance =

self . balances . setter (from) ; let old_sender_balance = sender_balance . get ( ) ; if old_sender_balance < value { return

Err (Erc20Error :: InsufficientBalance (InsufficientBalance

{ from , have : old_sender_balance , want : value , })) ; } sender_balance . set (old_sender_balance - value) ;

// Increasing receiver balance let

mut to_balance =

self . balances . setter (to) ; let new_to_balance = to_balance . get ( )

+ value ; to_balance . set (new_to_balance) ;

// Emitting the transfer event evm :: log (Transfer

{ from , to , value }) ; Ok (( )) }

/// Mints value tokens to address pub

fn

mint (& mut

self , address :

Address , value :

U256)

->

Result < ( ) ,

Erc20Error

{ // Increasing balance let

mut balance =

self . balances . setter (address) ; let new_balance = balance . get ( )

+ value ; balance . set (new_balance) ;

// Increasing total supply self . total_supply . set (self . total_supply . get ( )

+ value) ;

// Emitting the transfer event evm :: log (Transfer

{ from :

Address :: ZERO , to : address , value , }) ;

Ok (( )) }

/// Burns value tokens from address pub

fn

burn (& mut

self , address :

Address , value :

U256)

->

Result < ( ) ,

Erc20Error

{ // Decreasing balance let

mut balance =

self . balances . setter (address) ; let old_balance = balance . get ( ) ; if old_balance < value { return

Err (Erc20Error :: InsufficientBalance (InsufficientBalance

{ from : address , have : old_balance , want : value , })) ; } balance . set (old_balance - value) ;

// Decreasing the total supply self . total_supply . set (self . total_supply . get ( )

- value) ;

// Emitting the transfer event evm :: log (Transfer

{ from : address , to :

Address :: ZERO , value , }) ;

Ok (( )) } }

// These methods are external to other contracts // Note: modifying storage will become much prettier soon

[public]

impl < T :

Erc20Params

Erc20 < T

{ /// Immutable token name pub

fn

```
name ( )
->
String
{ T :: NAME . into ( ) }
/// Immutable token symbol pub
fn
symbol ( )
->
String
{ T :: SYMBOL . into ( ) }
/// Immutable token decimals pub
fn
decimals ( )
->
u8
{ T :: DECIMALS }
/// Total supply of tokens pub
fn
total_supply ( & self )
->
U256
{ self . total_supply . get ( ) }
/// Balance of address pub
fn
balance_of ( & self , owner :
Address )
->
U256
{ self . balances . get ( owner ) }
/// Transfers value tokens from msg::sender() to to pub
fn
transfer ( & mut
self , to :
Address , value :
U256 )
->
Result < bool ,
```

Erc20Error

{ self . _transfer (msg :: sender ( ) , to , value) ? ; Ok (true) }

/// Transfers value tokens from from to to /// (msg::sender() must be able to spend at leastvalue tokens from from) pub

fn

transfer_from (& mut

self , from :

Address , to :

Address , value :

U256 ,)

->

Result < bool ,

Erc20Error

{ // Check msg::sender() allowance let

mut sender_allowances =

self . allowances . setter (from) ; let

mut allowance = sender_allowances . setter (msg :: sender ( )) ; let old_allowance = allowance . get ( ) ; if old_allowance < value { return

Err (Erc20Error :: InsufficientAllowance (InsufficientAllowance

{ owner : from , spender :

msg :: sender ( ) , have : old_allowance , want : value , })) ; }

// Decreases allowance allowance . set (old_allowance - value) ;

// Calls the internal transfer function self . _transfer (from , to , value) ? ;

Ok (true) }

/// Approves the spenditure of value tokens of msg::sender() to spender pub

fn

approve (& mut

self , spender :

Address , value :

U256)

->

bool

{ self . allowances . setter (msg :: sender ( )) . insert (spender , value) ; evm :: log (Approval

{ owner :

msg :: sender ( ) , spender , value , }) ; true }

/// Returns the allowance of spender on owner's tokens pub

fn

allowance (& self , owner :

Address , spender :

Address)

->

U256

{ self . allowances . getter (owner) . get (spender) } }

lib.rs

// Only run this as a WASM if the export-abi feature is not set.

![cfg_attr(not(any(feature =

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

crate

alloc ;

// Modules and imports mod

erc20 ;

use

alloy_primitives :: { Address ,

U256 } ; use

stylus_sdk :: { msg , prelude :: * } ; use

crate :: erc20 :: { Erc20 ,

Erc20Params ,

Erc20Error } ;

/// Immutable definitions struct

StylusTokenParams ; impl

Erc20Params

for

StylusTokenParams

{ const

NAME :

& 'static

str

=

"StylusToken" ; const

SYMBOL :

& 'static

str

=

"STK" ; const

DECIMALS :

u8

=

18 ; }

// Define the entrypoint as a Solidity storage object. The sol_storage! macro // will generate Rust-equivalent structs with all fields mapped to Solidity-equivalent // storage slots and types. sol_storage!

{

[entrypoint]

struct

StylusToken

{ // Allows erc20 to access StylusToken's storage and make calls

[borrow]

Erc20 < StylusTokenParams

erc20 ; } }

[public]

[inherit(Erc20)]

impl

StylusToken

{ /// Mints tokens pub

fn

mint (& mut

self , value :

U256)

->

Result < ( ) ,

Erc20Error

{ self . erc20 . mint (msg :: sender ( ) , value) ? ; Ok (( )) }

/// Mints tokens to another address pub

fn

mint_to (& mut

self , to :

Address , value :

U256)

->

Result < ( ) ,

Erc20Error

{ self . erc20 . mint (to , value) ? ; Ok (( )) }

/// Burns tokens pub

fn

burn (& mut

self , value :

U256)

->

Result < ( ) ,

Erc20Error

{ self . erc20 . burn (msg :: sender ( ) , value) ? ; Ok (( )) } }

Cargo.toml

[package] name

=

"stylus_erc20_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 Gas metering Next Erc721