Oracle WebAssembly (Owasm)

Oracle WebAssembly, or Owasm for short, is Band Protocol's Domain Specific Language (DSL) on top of the Rust programing language for writingoracle scripts to be used in the BandChain ecosystem. The Owasm library consists of two primary modules:owasm/oei andowasm/ext .

- Owasm/OEI
-
  - The OEI modules defines a set of functions that are part of the Owasm Oracle Environment Interface. These functions are then accessible to an oracle script during its execution. The complete list of functions and implementation can be foundhere
- .
- Owasm/Ext
-
  - The Owasm extension module provides a convenient way to write oracle scripts that connect to various public APIs. Examples of these are functions to calculate the mean, median, and majority values from the validator's reported results, which can be used during the aggregation phase of an oracle script. The full list of functions and its implementation can be foundhere
- .

Usage

To illustrate an example usage of the Owasm library, we will be using the example below. The code is based off an oracle script for retrieving the price of a stock.

use

obi :: { OBIDeserialize ,

OBISchema ,

OBISerialize } ; use

owasm :: { execute_entry_point , ext , oei , prepare_entry_point } ;

[derive(OBIDeserialize, OBISchema)]

struct

Input

{ symbol :

String , multiplier :

u64 , }

[derive(OBISerialize, OBISchema)]

struct

Output

{ px :

u64 , }

[no_mangle]

fn

prepare_impl (input :

Input)

{ // Asking data source #14 for asset price oei :: ask_external_data (14 , 1 ,

& input . symbol . as_bytes ( )) ; }

[no_mangle]

fn

execute_impl (input :

Input)

->

Output

{ let avg :

f64

=

ext :: load_average (1) ; Output

{ px :

(avg * input . multiplier as

f64)

as

u64

} }

prepare_entry_point! (prepare_impl) ; execute_entry_point! (execute_impl) ; The script starts off by defining the input and output structs. In this case, the input comprises of the stock ticker/symbol (string) and the multiplier we want to multiply the stock's price by (u64). On the other hand, the output is simply the price of the stock multiplied by the multiplier, returned as au64 value.

Once the structs and types of both input and output have been determined, we move on to defining thepreparation andexecution phases of the oracle script, defined byprepare_impl andexecute_impl , respectively.

In order to call these functions, we need to pass appriopriate input values and make the function calls. To do so, oracle script writer can use our macros defined inmacros.rs , also shown below. The aim of these macros is to reduce the load of the script writer by handling the work of retrieving the calldata, deserializing it, and using it to construct the appropriate input struct for them.

[macro_export]

macro_rules! prepare_entry_point { (name : ident)

=>

{

[no_mangle]

pub

fn

prepare ( )

{ name (OBIDeserialize :: try_from_slice (& oei :: get_calldata ( )) . unwrap ( )) ; } } ; }

[macro_export]

macro_rules! execute_entry_point { (name : ident)

=>

{

[no_mangle]

pub

fn

execute ( )

{ oei :: save_return_data (& name (OBIDeserialize :: try_from_slice (& oei :: get_calldata ( )) . unwrap ( )) . try_to_vec ( ) . unwrap ( ) ,) ; } } ; } The last two lines of the oracle script above shows the macros in action.

Preparation phase

Theprepare_impl function takes the previously-defined input struct as an argument. The function then has only one main task; calling theask_external_data function inoei module.

In case extra information from relating oracle request is needed for implementing some logic before callingask_external_data , here are functions and their details fromoei module which can be used inprepare_impl .

- get_ask_count()
- returnsi64
-
  - Returns the number of validators to asked to report for this oracle request.
- get_min_count()
- returnsi64
-
  - Returns the minimum number of data reports as specified by the oracle request.
- get_calldata()
- returnsVec
-
  - Returns the raw calldata as specified when the oracle request is submitted.
-
  - Note:
-
  - This function is already called in macros when preparing the input.
- ask_external_data(eid: i64, did: i64, calldata: &[u8])
-
  - Takes in external id, data source id, and data in bytes.
-
  - Issues a new raw request to the host environment using the specified data source ID and calldata, and assigns it to the given external ID.

Execution phase

Theexecute_impl function takes in the input type as an argument as well, but also returns the output struct type. It then starts by computing the final value of the request through callingload_average function from theext module and proceed to use the computed average to construct and return the appropriate output struct.

Below is the list of functions fromoei that can be called inexecute_impl .

- get_ask_count()
- returnsi64
-
  - Returns the number of validators to asked to report for this oracle request.
- get_min_count()
- returnsi64
-
  - Returns the minimum number of data reports as specified by the oracle request.
- get_ans_count()

- returnsi64
-
  - Returns the number of validators that report data to this oracle request. Must only be called during execution phase.
- get_calldata()
- returnsVec
-
  - Returns the raw calldata as specified when the oracle request is submitted.
-
  - Note:
-
  - This function is already called in macros when preparing the input.
- save_return_data(data: &[u8])
-
  - Saves the given data as the result of the oracle execution. Must only be called during execution phase and must be called exactly once.
- get_external_data(eid: i64, vid: i64)
- returnsResult
-
  - Returns the data reported from the given validator index for the given external data ID. Result is OK if the validator reports data with zero return status, and Err otherwise. Previous On-chain Payment Next Introduction