Oracle WebAssembly (Owasm)

Oracle WebAssembly, or Owasm for short, is Band Protocol's Domain Specific Language (DSL) for writingoracle scripts to be used in the BandChain ecosystem.

Owasm Library Structure

The Owasm library is separated intoowasm-vm andowasm-kit for different use cases.

Owasm-vm

The Owasm-vm library implements a wasmer virtual machine runtime and logic for the on-chain execution of oracle scripts.

Owasm-kit

Owasm-kit/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.

OEI Module Functions

get_ask_count()

GetAskCount returns the number of validators asked to work for this oracle query.

Returns:

- ask_count: i64
-
    - The requested validator count.

get_min_count()

GetMinCount returns the minimum number of validators to move to aggregation phase.

Returns:

- min_count: i64
-
    - The sufficient validator count.

get_ans_count()

GetAnsCount returns the number of validators that submit answers. Zero during preparation.

Returns:

- ans_count: i64
-
    - The received validator count.

get_prepare_time()

GetPrepareTime returns the time ofpreparation phase was run.

Returns:

- timestamp: i64

-
    - The block timestamp during the preparation phase.

get_execute_time()

GetExecuteTime returns the time ofaggregation phase. Zero during preparation.

Returns:

- timestamp: i64
-
    - The block timestamp during the aggregation phase.

get_calldata()

GetCalldata returns the raw calldata as specified when the oracle request is submitted.

Returns:

- calldata: Vec
- : - The calldata in bytes.

save_return_data(&[u8])

SaveReturnData saves the given data as the result of the oracle execution. Must only be called during execution phase and must be called exactly once.

Parameters:

- data: &[u8]
-
    - The return data in bytes.

ask_external_data(i64, i64, &[u8])

AskExternalData 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. Must only be called during preparation phase.

Parameters:

- eid: i64
-
    - The external id to be assigned for the raw data source request.
- did: i64
-
    - The data source id to be requested.
- calldata: &[u8]
-
    - The calldata in bytes to be used in adid
- data source execution

get_external_data(i64, i64)

GetExternalData 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. Must only be called during execution phase.

Parameters:

- eid: i64
-

    - The external id of the data source execution to be returned.
- vid: i64
-
    - The validator index for the given external id.

Returns:

- result: Result
-
    - The data reported in String format from the given validator index for the given external data id.

Owasm-kit/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.

load_input\(i64)

LoadInput returns an iterator of raw data points for the given external id, parsed into the parameterized type. Skip data points with nonzero status or cannot be parsed.

Parameters:

- eid: i64
-
    - External id.

Returns:

- iterator: Iterator
-
    - Iterator of raw data points.

load_average\(i64)

LoadAverage returns the average value of the given external id, ignoring unsuccessful reports.

Parameters:

- eid: i64
-
    - External id.

Returns:

- result: Option
-
    - Average value of valid data points.

load_median\(i64)

LoadMedian returns the median value of the given external ID, ignoring unsuccessful reports.

Parameters:

- eid: i64
-
    - External id.

Returns:

- result: Option
-

- Median value of valid data points.

load_majority\(i64)

LoadMajority returns the majority value of the given external ID, ignoring unsuccessful reports.

Parameters:

- eid: i64
-
  - External id.

Returns:

- result: Option
-
  - Median value of valid data points.

The full 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 that retrieves the price of a stock.

use

obi :: { OBIDecode ,

OBIEncode ,

OBISchema } ; use

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

[derive(OBIDecode, OBISchema)]

struct

Input

{ symbol :

String , multiplier :

u64 , }

[derive(OBIEncode, OBISchema)]

struct

Output

{ px :

u64 , }

[no_mangle]

fn

prepare_impl (input :

Input)

{ oei :: ask_external_data (19 ,

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 (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 input and output structs and types have been defined, we move on to define the preparation and execution phases the oracle script, defined byprepare_impl andexecute_impl , respectively.

Theprepare_impl function takes the previously-defined input struct as an argument. The function itself then only peforms one task; calling theask_external_data function in theoei module. This call toask_external_data is then caught and ultimately resolved by BandChain throughexec_env.go .

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

Once we have defined the functions for both stages of the oracle script's execution, we need to pass in the appriopriate input values and actually 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 (OBIDecode :: 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 (OBIDecode :: 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. Previous Oracle Binary Encoding (OBI) Next Remote Data Source Executor