Creating an Oracle Script

In this section, we will take a look at how to create an oracle script.

Prerequisites

Rust Installation

While there are many ways to install Rust on your system. The official and recommended way to install Rust is using Rustup
.

Rustup installsrustc ,cargo ,rustup and other standard tools to Cargo's bin directory. On Unix it is located atHOME/.cargo/bin and on Windows at%USERPROFILE%.cargo\bin . This is the same directory that cargo install will install Rust programs and Cargo plugins.

After installing Rust you can check the current version by typingrustc --version orrustc -V on your terminal to verify the success of the installation.

Note: If wasm32-unknown-unknown hasn't been added as a target, you can add it using the command below.

```
rustup target add wasm32-unknown-unknown
```

Writing the Oracle Script

File structure

Let's start by creating a Rust directory structure like in the example below.

```
. ├── hello_world ├── Cargo.toml └── src └── lib.rs
```

Adding Dependencies

AsCargo.toml is the manifest file for Rust's package manager: Cargo, this file contains metadata such as the name, version and dependencies of the package. By default, Cargo checks dependencies on crates.io. Therefore, when adding a crate, we only need to add the crate name and version to theCargo.toml .

When creating an oracle script, two main dependencies are required:owasm-kit andobi .

An example is shown below:

```
[package] name = "hello-world"

version = "0.1.0" authors = ["Band Protocoldev@bandprotocol.com"] edition = "2018"

[lib] crate-type = ["cdylib"]

[dependencies] owasm-kit = { version = "0.1.13" } obi = { version = "0.0.2" }
```

Writing the Oracle Script

As mentioned in theintroduction , an oracle script execution flow can be categorized into two main phases, the preparation phase and the execution phase. However, we also do need to define the oracle scripts input and outputs.

Input/Output

An oracle script's input and output can be defined in a struct. In the example below, we can see that this specific oracle scripts takes in an inputrepeat as au64 and returns an outputresponse as astring

[derive(OBIDecode, OBISchema)]

struct

Input

{ repeat :

u64 , }

[derive(OBIEncode, OBISchema)]

struct

Output

{ response :

String , }

Preparation Phase

The function below shows an example preparation phase for requesting data from data sourceD327 . AsD327 does not require any inputs, an empty byte will be passed. However, in other data sources that do require an input, the corresponding calldata should be sent instead.

const

DATA_SOURCE_ID :

i64

=

327 ; const

EXTERNAL_ID :

i64

=

0 ;

[no_mangle]

fn

prepare_impl (_input :

Input)

{ oei :: ask_external_data (EXTERNAL_ID ,

// The assigned external ID for this data source DATA_SOURCE_ID ,

// The data source to call by ID b"" ,

// Calldata to be sent to the data source) }

Execution Phase

The function below shows an example of the execution phase for the data received fromD327 . This example retrieves the data reports and duplicates the majority result of the data report$x$ times where$x$ is defined byrepeat as given in the input.

[no_mangle]

fn

execute_impl (input :

Input)

->

Output

{ let raw_result =

ext :: load_input :: < String

(EXTERNAL_ID) ;

// Raw results from the given external ID let result :

Vec < String

= raw_result . collect ( ) ; let majority_result :

String

=

ext :: stats :: majority (result) . unwrap ( ) ;

// Majority result Output

{ response : majority_result . repeat (input . repeat as

usize) , } }

lib.rs

use

obi :: { OBIDecode ,

OBIEncode ,

OBISchema } ; use

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

[derive(OBIDecode, OBISchema)]

struct

Input

{ repeat :

u64 , }

[derive(OBIEncode, OBISchema)]

struct

Output

{ response :

String , }

const

DATA_SOURCE_ID :

i64

=

327 ; const

EXTERNAL_ID :

i64

```
=

0 ;
```

[no_mangle]

```
fn

prepare_impl ( _input :

Input )

{ oei :: ask_external_data ( EXTERNAL_ID , DATA_SOURCE_ID , b"" , ) }
```

[no_mangle]

```
fn

execute_impl ( input :

Input )

->

Output

{ let raw_result =

ext :: load_input :: < String

      ( EXTERNAL_ID ) ; let result :

Vec < String

= raw_result . collect ( ) ; let majority_result :

String

=

ext :: stats :: majority ( result ) . unwrap ( ) ; Output

{ response : majority_result . repeat ( input . repeat as

usize ) , } }

prepare_entry_point! ( prepare_impl ) ; execute_entry_point! ( execute_impl ) ;
```

Compling the Oracle Script

To compile the oracle script, the following command can be run

RUSTFLAGS

'-C link-arg=-s' cargo build --release --target wasm32-unknown-unknown After the compilation is complete, the.wasm file can be found in the sub-directory:./target/wasm32-unknown-unknown/release/*.wasm .

More Examples

Below is another example of an oracle script that queries a token's total supply.

Query for token total supply

lib.rs

```
use
```

obi :: { OBIDecode ,

OBIEncode ,

OBISchema } ; use

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

[derive(OBIDecode, OBISchema)]

struct

Input

{ rpc :

String , to :

String , }

[derive(OBIEncode, OBISchema)]

struct

Output

{ total_supply :

String , }

const

DATA_SOURCE_ID :

i64

=

98 ; const

EXTERNAL_ID :

i64

=

0 ;

[no_mangle]

fn

prepare_impl (input :

Input)

{ oei :: ask_external_data (EXTERNAL_ID , DATA_SOURCE_ID , format! ("{} {}" , input . rpc , input . to) . as_bytes ( ) ,) ; }

[no_mangle]

fn

execute_impl (_input :

Input)

->

```
Output

{ Output

{ total_supply :

ext :: load_majority :: < String

        ( EXTERNAL_ID ) . unwrap ( ) , } }

prepare_entry_point! ( prepare_impl ) ; execute_entry_point! ( execute_impl )
```

Previous Introduction Next Deployment