Oracle Binary Encoding (OBI)

Oracle Binary Encoding (OBI) is a standard method for serializing and deserializing binary data in the BandChain ecosystem. Under the concept of Ethereum'sContract ABI Specification and Google'sProtoBuf , an OBI schema explains how a data object in any supported programming language can be encoded to and decoded from plain bytes.

OBI is designed with the following properties in mind:

- Compactness
- : OBI schema will be stored on-chain and passed around between blockchains. Thus, it is essential to keep the size of the schema specification tiniest.
- Simplicity & Portability
- : As a blockchain-agnostic protocol, OBI serialization and deserialization must be easy to implement in any environment. Consequently, complex platform-specific features are not supported.
- Readability
- : Lastly, OBI is intended to be used as a communication tool between oracle script creators and smart contract developers. It must be intuitive for readers to understand the OBI underlying objects from reading the schema.

Specification

An OBI schema is a non-self-describing binary serialization format of multiple objects. Some particular notes:

- An OBI schema consists of one or more individual schemas. In most cases, an OBI schema will consist of two individual schemas: the input type and the output type.
- bool is supported.
- 6 sizes (8-bit, 16-bit, 32-bit, 64-bit, 128-bit, and 256-bit) of signed and unsigned integers are supported. There are all serialized into big-endian bytes.
- Strings, bytes, vectors are serialized with their length as u32 first, followed by their contents.
- Structs are serialized field by field in the declaration order.

Backus–Naur Form (BNF) Grammar Specification

Below is theBackus–Naur form (BNF) grammar of an OBI schema.

< obi_schema

::

< obi_schema

|

< obi_schema

"/"

< indv_schema

< indv_schema

::

< int_schema

|

< uint_schema

|

< string_schema

| < bytes_schema

|

< vector_schema

|

< struct_schema

< bool_schema

::

"bool" < int_schema

::

"i8"

|

"i16"

|

"i32"

|

"i64"

|

"i128"

|

"i256" < uint_schema

::

"u8"

|

"u16"

|

"u32"

|

"i64"

|

"u128"

|

"u256" < string_schema

::

"string" < bytes_schema

::

"bytes" < vector_schema

::

"["

< indv_schema

"]" < struct_schema

::

"{"

< struct_fields

"}" < struct_fields

::

< struct_member

|

< struct_fields

","

< struct_member

< struct_member

::

< identifier

":"

< indv_schema

Pseudocode Implementation

Below is an examplepseudocode implementation of OBI schema declaration and the corresponding serializing function in a somewhat broken function language :P. The deserialization function is essentially the inverse of the serialization function.

(*An individual schema consists of 6 possible cases.*) type indv_schema := | Bool (bool) | Int (int) | Uint (int) | String | Bytes | Vector (indv_schema) | Struct ([(string , indv_schema)])

(*An OBI schema is essentially a list of individual schemas.*) type obi_schema :=

[indv_schema]

(*Encode serializes the given object into bytes.*) let encode (s : indv_schema)

(o :

object)

:= match s with | Bool (sz)

=> be_unsigned_encode (o , sz) | Int (sz)

=> be_signed_encode (o , sz) | Uint (sz)

=> be_unsigned_encode (o , sz) | String => be_unsigned_encode (len (o) ,

32)

++ bytes_of_string (o) | Bytes => be_unsigned_encode (len (o) ,

32)

++ o | Vector (s)

=> be_unsigned_encode (len (o) ,

32)

++ concat (map o (encode s)) | Struct (fs)

=> concat (map fs (fun

(f , s)

=> encode s o [f]))

OBI Schema Examples

Below is an example OBI schema of an oracle script to fetch a cryptocurrency price, which is then multiplied by a specific multiplier. The OBI itself schema consists of two internal schemas, one for the inputs to the oracle script and the other for the output.

- The input consists with two fields: a string symbol and a u64 multiplier.
- The output consists with two fields: a u64 final price and a vector of struct each has string name and u64 timestamp.

Compact OBI representation...

{ symbol : string , multiplier : u64 } / { price : u64 , sources : [{ name : string , time : u64 }] }

Prettified OBI representation...

{ symbol : string , multiplier : u64 } / { price : u64 , sources :

[{ name : string , time : u64 }] }

Example Object Serialization

{ "symbol" :

"BTC" , "multiplier" :

1000000000 } 0x0000000342544300000000 3b9aca00 ^ ^ ^ |

| +- 64 -bit be encode of 1000000000 is 0x000000003b9aca00 | +------- "BTC" data is encoded as 0x425443 +---------- 32 -bit be encode of length 3 is 0x00000003

BTC is a string with a length of 3

{ "price" :

9268300000000 , "sources" :

[{ "name" :

"CoinGecko" , "time" :

1590305341 } , { "name" :

"CryptoCompare" , "time" :

1590305362 }] }
0x0000086df1baab000000000200000009436f696e4765636b6f000000005eca223d0000000d43727970746f436f6d70617265000000005eca2252
^ ^ ^ ^ ^ ^ ^ ^ |

|

|

|

|

|

| +- 64 -bit be encode of 1590305362 is 0x000000005eca2252 |

|

|

|

|

| +- "CryptoCompare" data is encoded as 0x43727970746f436f6d70617265 |

|

|

|

| +- 32 -bit be encode of length 13 is 0x0000000d

CryptoCompare length is 13

|

|

|

| +- 64 -bit be encode of 1590305341 is 0x000000005eca223d |

|

| +- "CoinGecko" data is encoded as 0x436f696e4765636b6f |

| +- 32 -bit be encode of length 9 is 0x00000009

CoinGecko length is 9

| +------- 32 -bit be encode of length 2 is 0x00000002

sources has a value that is an array with a length of 2

+---------- 64 -bit be encode of 9268300000000 is 0x0000086df1baab00

Reference Implementations

OBI serialization libraries are being actively developed in multiple programming languages. Head over to BandChain's OBI module to see all currently available implementations. Previous Example Use Cases Next Oracle WebAssembly (Owasm)