Author: S.G.
Title : Specification DHI-4D 45 4D 4F
(DigitalHorizonImplementation–M E M O)

1. The Idea

We decided to add a Memo field on the Ethereum based networks in case native protocol doesn't support them. Tron and several other blockchain protocols implemented them and we found the function quite useful; solving a lot of problems using this networks.

2. Why not implement on the network itself?

We found it complicated discussing with communities to get new ideas and proposals 'accepted' and maybe somewhere in some point in the future implemented and usable on the network. Besides the bureaucracy it would require at least a softfork of the network. Our review of the protocol source and tests we did enables us to use the feature using the existing structure without causing incompatibilities or updates in any way. It can be realized by simply adding a layer of data where it's not used yet and interpret them in the way we like: As a transaction memo. Anyone who wants to participate can do with simple changes on the code and others who doesn't want, or care, can just ignore it like they do right now. There is no need to add or change anything to get such functionality implemented on the current chains.

3. So, how do it?

We just append the memo to the (input) data section of the transaction. By specification the field is used:

- To deploy a smart contract
  Transactions deploying a smart contract are send to 'null'. This cases won't carry a memo.

- To interact with a smart contract.
  The EVM parses only the data specified and ignores any additional data. Appending data after the payload meant as function parameters for the contract is ignored and not affecting the SC.

4. Let’s define some standards

4.1. Prefix

We use 0x4D454D4F3A (hex(“MEMO:”)) as prefix to start a memo. Using a five byte prefix serves several purposes:

a) In case somebody else uses the data field for custom data which is not meant as a memo in our way we can ignore it.

b) Keep other developers code compatible. A lot of parsesr for transactions out there assume if data is present that they deal with a smart contract and expect the first for bytes as function signature. Using a 4 bytes prefix ensures that this parsers have the data present they expect and they will just assume some custom function which they don’t know their ABI.

c) Just using 4 bytes as prefix could match a real existing signature. Till time of writing there’s no function having 0x4d454d4f so far but it’s possible. To distinct a memo from a real signature with the same footprint we add an additional byte. Chances having a function with same signature and first byte of its padded first parameter being 0x3A is so low that we take the risk of interpreting a SC function and its data for a memo when it’s not.

4.2 Padding

We pad the memo to 32 byte sized chunks to the end, using 0x00 as padding character:

a) Keep other developers code compatible. Parameters for smart contracts are padded to 32 bytes. Parsers not implementing our memo may crash when data is formatted different than expected. So we keep the format the ‘Ethereum way’.

4.3 Encoding

Memo is UTF-8 encoded and this value is interpreted as of bytes type and encoded further.

4.4 Max Length

Hard limit is the max. allowed size of a transaction on chain of course. We use a soft limit of 30 chunks = 30*32=960 bytes.

5. Proof of concept:

Sending a token requires SC operation. Here we're adding a memo to a token transfer to show the added data is not affecting the SC operation.

We appended a memo with content:

00020126580014BR.GOV.BCB.PIX01364a35efba-0efd-4006-b034-a0f826451dcd5204000053039865405 13.005802BR5925Francisco Rocelo Bezerra 6009SAO PAULO61080540900062240520G8TvPTkz6ocQjTi26bqe630424B7:13

0xde2cc28395ca1bcf9acf3ed49103acf920632b0c937cf47215dbefacc9d0e681