# Efficient storage of arbitrary data in a Bitcoin-derived blockchain with a large OP_RETURN

**Document version 0.4**

Fredrick R. Brennan
copypaste@kittens.ph

Ronald Watkins
admin@susukino.com

September 11, 2018

# 1 Introduction

Storage of arbitrary data in blockchains has been around since the very early days of Bitcoin. This trend could even be said to hark back to Satoshi Nakamoto, who used the `coinbase` field of the Bitcoin genesis block to include the text "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks[1]." Since then, arbitrary data has found its way into the blockchain via numerous routes—the outputs of a transaction, the `OP_RETURN` field, even other tricky methods such as using P2PKH transactions.

A notable recent entry in this trend is the Memo protocol.[2] Memo implements a social network via the extended `OP_RETURN` size of 220 bytes found in Bitcoin Cash.[3]

This paper proposes a system, the *Storage Utility Memory Object* (**SUMO**), that could be used to efficiently store data in the Susucoin blockchain and/or other similar blockchains.

## 1.1 Goals

When this standards document was produced, the following goals were taken into account:

1. **Efficiency.** Every byte counts on the blockchain and the goal of this project is to make them count. Convenience for programmers

[1]For more examples of arbitrary data storage in the blockchain, see `http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html`.

[2]`https://memo.cash/about`

[3]Described at `https://memo.cash/protocol`. Similar to a SQL database's `VARCHAR` limit of 255, the limits in the "Value" column of the table are hard limits.

matters, but small footprints matter more, so everything possible was done to get a smaller size;

2. **Ease of implementation.** The standard must be easy to implement on a variety of platforms, including web applications and mobile apps;

3. **Extensibility.** It must be possible to extend the standard to support different compression methods, different data types, and different metadata.

SUMO's goal is to provide a protocol that can efficiently, easily, and extensibly store data into the blockchain in a permanent fashion.

# 2 The SUMO Protocol

SUMO is an extensible protocol that can adapt to the needs of the implementing blockchain.

## 2.1 Note About SUMO's Integers

Unless stated otherwise, all header elements are variable-length quantities (VLQs), known as `uintvars`. A `uintvar` is, at its most basic, a signed 8-bit integer that uses its sign bit to denote that another integer follows. This means that all positive values that would fit into an unsigned integer (0-127) can still be stored in one byte, but an integer of any size can be stored, even integers larger than those supported by the CPU architecture.

## 2.2 Header

SUMO's header is variable-length, the most expected case being two bytes: magic number and action.

The header is as follows:

| Version | Flags | Message number | Part | Total | Reference |
|---|---|---|---|---|---|

$$\text{Version}\;\text{Flags}\;\underbrace{\text{Message number}\;\text{Part}\;\text{Total}}_{\text{if multipart}}\;\underbrace{\text{Reference}}_{\substack{\text{if reference}\\ \text{\& first or only part}}}$$

### 2.2.1 Version

Version is a `uintvar`. The first version will be `0x01`.

### 2.2.2 Flags

The following flags are defined as a bitmask of a byte; 0 is false, 1 is true:

- Bit 1: Does another byte follow this one?[1]
- Bit 2: Is payload a reference to other arbitrary data elsewhere on the blockchain?
- Bit 3: Is payload multipart?
- Bit 4–5: At the discretion of implementors; could be used to add more available compression algorithms, or for other purposes;
- Bit 6–8: A three-bit integer that represents a compression algorithm.

#### Available Compression Algorithms

- 000: Uncompressed;
- 001–111 (1–7): At the discretion of implementors.

As all blockchains are different, it is difficult to predict the type of data that would be stored. Generally, because short strings are being stored, entropy encoders are preferrable; Zstandard is an example of an entropy encoder. If mostly ASCII or Latin-1 text will be stored, Shoco is a good choice; for the string "There was an old man in London named Trent," it achieves a compression ratio of 26% as of June 28, 2018.[2].

It is quite likely implementors will want to store multiple different types of data. As long as all clients agree on which three-bit integer refers to which deflation method, this is not a problem. Encoders need not necessarily know how to encode all the methods, but decoders

[1] This could be used to implement more options, like adding other data to the header at the discretion of implementors.

[2] From the Shoco demo: `http://ed-von-schleck.github.io/shoco/`

must. As the input strings are short, a competent encoder will try to compress the input with all available options to find the best one for the data.

For example, if you want to store both Japanese and Chinese text, Zstandard could be trained against the input of Japanese and Chinese Wikipedia database dumps, stripped of extra characters, to produce two dictionaries. `Japanese.dict` + Zstandard could be assigned to compression Method 1, and `Chinese.dict` + Zstandard assigned to Method 2, with methods 3–7 reserved for future assignment. If you later want your blockchain to support the storage of compressed PNG files, you could add a PNG compression algorithm to the reserved Method 3.

### 2.2.3 Message Number

Message number is a `uintvar`. This is used to differentiate between different multipart messages sent from the same address. All parts of a message must be sent from the same address. As this is a `uintvar`, if more than 127 messages are sent from one address, there will be two bytes.

### 2.2.4 Part & Total

Part and total are `uintvar`. They are used in multipart messages. Except for large amounts of data, these will usually be one byte. With a maximum `OP_RETURN` of 256 bytes, and a header of five bytes, a payload of $\simeq$ 32KB can be stored with one-byte part and total. It is up to clients to decide how much arbitrary data they would like to support in their implementation of the SUMO Protocol.

### 2.2.5 Reference

A reference is a 32-byte transaction ID, used if the input data is a concatenate to other data from the same subset. Multiple transactions can be referenced to by delimiting them with byte 0x1D. In the payload, use 0x1A to insert a reference to the leftmost non-nothing value.

For multiple transactions, put an ASCII digit after the transaction, like 0x1A 0x32 (ASCII '2'). The maximum amount of data that can be referenced to depends on the implementation of the parent blockchain's maximum `OP_RETURN` size.

## 2.3 Payload

The payload contains the non-header data to be stored. The payload can be either compressed or uncompressed, depending on the header flags. This is to prevent the compression algorithm from doing more harm than good because compression ratios can sometimes be greater than 1.0. Since a compression algorithm does not signify the type of data being stored, a compression algorithm made for one type of data might compress a different kind of data more efficiently and may be chosen by the encoder. Implementing clients must mark the type of data in the payload itself. For example, if you need to disambiguate UTF-8 text, ASCII text and SHIFT_JIS, you could 1) use a byte-order mark (BOM) for UTF-8; 2) rely on a heuristic guarantee that all bytes in the 0-127 range represent ASCII text; and 3) consider all other input to be SHIFT_JIS.

## 2.4 Security

Security is a concern with all software. Even though the SUMO Protocol was created to be easily implemented while respecting the need to save as much space as possible, it still allows the use of compression, and is therefore vulnerable to zip bomb attacks. Care should be taken by implementing clients to limit the amount of RAM they are willing to spend, e.g., `ZSTD_DCtx_setMaxWindowSize`.

As the SUMO Protocol is variable-length, frequent sanity checks are highly encouraged. A memory-safe compiled language like Rust or Haskell could be used and then included in other programs via foreign function interfaces (FFIs).

# 3 Examples

The SUMO Protocol could be implemented for various applications, such as community migrations to alternative online platforms focused on free speech or the free exchange of ideas among individuals from countries with authoritarian regimes or in a world with increasingly powerful multinational corporations who believe it is their job to police speech.

An example of a hypothetical three-bit compression header flag could be:

- 000: Uncompressed;
- 001: Zstandard with a dictionary trained against SHIFT_JIS;
- 010: Shoco, trained against English;
- 011: Shoco, trained against Spanish;
- 100: Zstandard with a large dictionary trained against numerous Wikipedia articles;
- 101: Brotli (LZ77);
- 110: Unused;
- 111: Unused.

## 4 Final Remarks

Free speech is under siege from misguided policymakers passing speech-restrictive legislation; special-interest groups and state actors employing agents to disrupt the operations of organized activists and concerned citizens; and giant multinational corporations trying to moderate speech out of their platforms using restrictive terms and conditions as a cover for the deep-seated political biases of top executives. As such, by enabling the free exchange of thoughts, opinions, and ideas through the addition of data to a blockchain, the SUMO Protocol boasts various speech-protective applications that could help further individual liberty.