32/Z85 - ZeroMQ Base-85 Encoding Algorithm

This document specifies Z85, a format for representing binary data as printable text. Z85 is a derivative of existing Ascii85 encoding mechanisms, modified for better usability, particularly for use in source code.

- Name: rfc.zeromq.org/spec:32/Z85
- Editor: Pieter Hintjens <ph@imatix.com>

Preamble

This Specification is a free and open standard and is governed by the Digital Standards Organization's Consensus-Oriented Specification System.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Problem Statement

Use the zeromq-dev list to discuss this specification.

This specification has been deployed to real users (stable).

Set the state of this specification

- New specification (raw)
- Has at least one implementation (draft)

When representing binary values in code and in text files (such as configuration data), the developer must choose a printable representation. The simplest choice is Base16, where each byte is printed as two hexadecimal values.

For better efficiency than Base16, the developer might choose Base64. This is a common choice but problematic because it has more than a dozen variants. It works poorly with binary data since it works on three-byte chunks, whereas most binary data is chunked to 4 or 8 bytes. Base64 implementations are therefore more complex than they might be, and not necessarily interoperable.

A more logical encoding is Ascii85, which works on 4-byte chunks. However it is not string-safe, so cannot be used cleanly in source code, XML, JSON, and so on.

Our ideal encoding is designed for 4-byte chunks like Ascii85, but is safe for strings, and removes all ambiguity about whether or not implementations will interoperate. Z85 is thus designed to be more compact than Base16, more reliable than Base64, and more usable than Ascii85.

The specific goals of this specification are:

- To provide the most efficient textual representation possible.
- To be easy to use in source code, when enclosed in double or single quotes.
- To be safe to pass on the command line, when enclosed in single quotes.
- To be easy to implement in any programming language.

Additionally, binary zero strings should be easily visible (we achieve this by mapping zero to the ASCII character '0').

Formal Specification

A Z85 implementation takes a binary frame and encodes it as a printable ASCII string, or takes an ASCII encoded string and decodes it into a binary frame.

The binary frame SHALL have a length that is divisible by 4 with no remainder. The string frame SHALL have a

- Being replaced by newer specs (legacy)
- Replaced, no longer used (retired)
- Abandoned before becoming stable (deleted)

Specifications

39/ZWS - ZeroMQ WebSocket Protocol - *Raw*

37/ZMTP - ZeroMQ Message Transport Protocol - *Raw*

40/XRAP - Extensible Resource Access Protocol - *Draft*

38/ZMTP GSSAPI Security Mechanism - *Draft*

36/ZeroMQ Realtime Exchange Protocol - *Draft*

35/FILEMQ - File Message Queuing Protocol - *Draft*

34/SRPZMQ Authentication

length that is divisible by 5 with no remainder. It is up to the application to ensure that frames and strings are padded if necessary.

The encoding and decoding SHALL use this representation for each base-85 value from zero to 84:

```
 0 -  9:  0 1 2 3 4 5 6 7 8 9
10 - 19:  a b c d e f g h i j
20 - 29:  k l m n o p q r s t
30 - 39:  u v w x y z A B C D
40 - 49:  E F G H I J K L M N
50 - 59:  O P Q R S T U V W X
60 - 69:  Y Z . - : + = ^ ! /
70 - 79:  * ? & < > ( ) [ ] {
80 - 84:  } @ % $ #
```

To encode a frame, an implementation SHALL take four octets at a time from the binary frame and convert them into five printable characters. The four octets SHALL be treated as an unsigned 32-bit integer in network byte order (big endian). The five characters SHALL be output from most significant to least significant (big endian).

To decode a string, an implementation SHALL take five characters at a time from the string and convert them into four octets of data representing a 32-bit unsigned integer in network byte order. The five characters SHALL each be converted into a value 0 to 84, and accumulated by multiplication by 85, from most to least significant.

Test Case

As a test case, a frame containing these 8 bytes:

```
+------+------+------+------+------+------+-----
| 0x86 | 0x4F | 0xD2 | 0x6F | 0xB5 | 0x59 | 0xF7
+------+------+------+------+------+------+-----
```

SHALL encode as the following 10 characters:

and Encryption Protocol - *Draft*

33/ZHTTP - HTTP Request-Reply over ZeroMQ - *Draft*

18/MDP - Majordomo Protocol 0.2 - *Draft*

17/ZDCF - ZeroMQ Device Configuration File - *Draft*

12/CHP - Clustered Hashmap Protocol - *Draft*

11/MTL - Message Transfer Layer - *Draft*

10/FLP - Freelance Protocol - *Draft*

9/TSP - Titanic Service Protocol - *Draft*

8/MMI - Majordomo Management Interface - *Draft*

6/PPP - Paranoid Pirate Protocol - *Draft*

5/ZDCF - ZeroMQ Device

```
+---+---+---+---+---+---+---+---+---+---+
| H | e | l | l | o | W | o | r | l | d |
+---+---+---+---+---+---+---+---+---+---+
```

Reference Implementation

A reference implementation in C is provided in the RFC repository at https://github.com/zeromq/rfc/blob/master/src/spec_32.c.

Security Considerations

Implementations MUST take care that buffers intended to receive encoded and decoded data are large enough. Clearly, a properly-crafted encoded string can create any arbitrary binary sequence, which makes Z85 an easy vector for attack into a badly-designed implementation.

Configuration File - *Draft*

32/Z85 - ZeroMQ Base-85 Encoding Algorithm - *Stable*

31/EXPAIR - ZeroMQ Exclusive Pair Pattern - *Stable*

30/PIPELINE - ZeroMQ Pipeline Pattern - *Stable*

29/PUBSUB - ZeroMQ Publish-Subscribe Pattern - *Stable*

28/REQREP - ZeroMQ Request-Reply Pattern - *Stable*

27/ZAP - ZeroMQ Authentication Protocol - *Stable*

26/CurveZMQ Authentication and Encryption Protocol - *Stable*

25/ZMTP CURVE Security Mechanism - *Stable*

24/ZMTP PLAIN Security

Mechanism - *Stable*

23/ZMTP - ZeroMQ Message Transport Protocol - *Stable*

22/C4.1 - Collective Code Construction Contract - *Stable*

21/CLASS - C Language Style for Scalability - *Stable*

20/ZeroMQ Realtime Exchange Protocol - *Stable*

19/FILEMQ - File Message Queuing Protocol - *Stable*

15/ZMTP - ZeroMQ Message Transport Protocol - *Stable*

7/MDP - Majordomo Protocol - *Stable*

4/ZPL - ZeroMQ Property Language - *Stable*

Show older specs

People who are watching this page:

- martin_sustrik
- Daniel Moore
- ;-) mgoetzke
- guido_g
- ;-) dannoy
- ;-) Martin Hurton
- ;-) salsudan
- ;-) robothor
- ;-) feamsr00
- ;-) Robert Varga
- ;-) Jan Vlcinsky
- ;-) rodgertq
- ;-) CzajNick
- ;-) Diego Duclos
- Olivier Lance
- ;-) goffer123
- ;-) hatbro
- ;-) binocarlos
- ;-) jalcine
- ;-) haarts

... and more

Watch: site | category | page