Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 3 October 2026

S. Morgan
Adligo Inc.

1 April 2026

Text Encoded Base 64 Numbers (Ten64)
draft-morgan-ten64-00

Abstract

Ten64 is a positional numeral system for representing numeric values with an alphabet of 64 characters (aka. radix/base 64). However, unlike most positional number systems, the characters are written in ascending (aka. big-endian, network-order) and NOT descending order. The main differences are the use of sextets (six bits) instead of octets (aka. bytes). Similar to hexadecimal, Ten64 can be used to create binary strings of arbitrary length. Ten64 can encode one or more Modern Western Numbers (aka. Arabic, Vedic), interpreting the characters respective composite big-endian binary as a one or more Modern Western Numbers.

The motivation for Ten64 is to encode numbers in a compact and human-readable format similar to Base58. However, Ten64 is designed to be optimized for use with numbers commonly used in identifiers, such as DIDs, DOIDs, IANA OIDs, UUIDs, dates, time(stamp)s, points, and more. Unlike Base58, and more like hexadecimal Ten64 aligns the Modern Western Numerals with the respective big-ending binary (i.e.; 0 → 0, 1 → 1, 2 → 11, etc). Finally, Ten64 provides a disk-based number encoding system for huge numbers and number streams.

Table of Contents

1. Introduction

This section is non-normative.

Alternative notations and bases have historically been explored to improve human-machine interfaces, ranging from early discussions on binary notations to modern concepts like Hexadecimal, and Bioctal. Ten64 builds on this history by using a 64-character alphabet composed of standard ASCII-7 / UTF-8 characters.

In addition to providing a solid manor to represent numbers using characters and their respective binary representations, Ten64 extends these representations with interpretation conventions. These interpretation conventions will likely be specified by some sort of schema system like EJCN (Extensible JSON Classification Notation) Schemas, JSON Schemas or XML Schemas.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document aligns with the formal IETF standardization procedures defined by the Internet Standards Process in BCP 9 [RFC2026].

2. Human-Readable Verses Human-Decipherable

In Ten64, we are targeting human-readable characters in the alphabet but do NOT intend to be human-decipherable. Our definition of these terms MAY be specific to this paper.

2.1. What is Human-Readable?

Our intended meaning of the term human-readable in Ten64 is that a human will be able to distinguish the characters from each other. We believe the best way of explaining this is with an example where one human would literally read the characters to another human over an audio-only telephone call. For example, the word Adligo is spelled out using a spelled-out phonetic alphabet.

Capital A, as in Apple.
Small d, as in dog.
Small l, as in lake.
Small i, as in indigo.
Small g, as in golf.
Small o as in otter.

This is clearly human-readable.

2.2. What is Human-Decipherable?

Our intended meaning of the term human-decipherable in Ten64 is that most humans would be able to decipher the character stream. Ten64 does NOT target human-decipherability. We consider human-decipherability to be out of scope and may target another ID/ RFC for this work. Ten64 targets computer-decipherability and will rely on computer algorithms to translate Ten64 into in-memory integer and decimal numbers. In particular;

* The Ten64 Integer Serialization Algorithm

* The Ten64 Decimal Serialization Algorithm

Although this is decipherable for some humans who are developers or software architects, this point might seem insignificant. For the vast majority of the population, it is significant. The point is that software tools will generally be used to decipher Ten64.

3. Special Characters Introduction

Ten64 does NOT use the Base64 RFC 4648 alphabet but a alphabet more in line with hexadecimal 0-9,a-k,$,m-z,A-H,+,J-N,!,P-Z, '@' and '_' along with some additional special characters most notably '#','.',',' and ';', '-', and the UNIX Line Feed 10 (0x0A in hexadecimal). It is designed to be human and machine-readable but is really designed to be optimized the reading and writing of numbers for streaming and storage computer systems. The $,+,!, @ and _ symbols were chosen because they human-readable. Theoretically, this system could also be used embed numbers into programming languages in the future. However, usage MAY require the explicit starting character pound '#', and explicit termination semicolon character ';'. It will use a big ending binary system as follows;

4. Special Characters and Sequence Details

```
#   Optional explicit beginning of #Ten64 binary section, use depending
    on context.
#.. The optional explicit beginning of a multiple line Ten64 sequence.
.   The Decimal, List or Number Space Separator
,   The Separator for Number Lists
;   Optional explicit end of #Ten64 sequence.
-   The negative indicator, and human-readable separator
    Whitespace Characters: including Line Feeds, Tabs,
       Spaces, Return Sequences, etc. MAY be included
       and MUST cause end of interpretation of the
       Ten64 sequence.
UNIX Line Feeds:
    UNIX line feeds 10 / hex 0x0A
    MAY be used to continue a number sequence when the number
    sequence starts with '#..'.

    Other Non Alphabet Characters: MAY be included
       and MUST cause end of interpretation of the
       Ten64 sequence.
```

5. The Ten64 Alphabet Mappings

Primary ASCII / UTF8	Modern Western Integer Value	Big-Ending Binary Sextet
0	0	000000
1	1	100000
2	2	010000
3	3	110000
4	4	001000
5	5	101000
6	6	011000
7	7	111000
8	8	000100
9	9	100100
a	10	010100
b	11	110100
c	12	001100
d	13	101100
e	14	011100
f	15	111100
g	16	000010
h	17	100010
i	18	010010
j	19	110010
k	20	001010

$	21	101010
m	22	011010
n	23	111010
o	24	000110
p	25	100110
q	26	010110
r	27	110110
s	28	001110
t	29	101110
u	30	011110
v	31	111110
w	32	000001
y	33	100001
x	34	010001
z	35	110001
A	36	001001
B	37	101001
C	38	011001
D	39	111001
E	40	000101
F	41	100101
G	42	010101
H	43	110101
+	44	001101

J	45	101101
K	46	011101
L	47	111101
M	48	000011
N	49	100011
!	50	010011
P	51	110011
Q	52	001011
R	53	101011
S	54	011011
T	55	111011
U	56	000111
V	57	100111
W	58	010111
X	59	110111
Y	60	001111
Z	61	101111
@	62	011111
_	63	111111

Table 1: Ten64 Alphabet Mappings

6. Use-Cases

6.1. Shortened Identifiers

The primary Use-Case is simply an upgrade of hexadecimal, where Ten64 provides a more succinct/compressed string of characters. In addition, we target the encoding of numbers (n in the *Text Encoded Base 64 Numbers* title). Generally, we are also targeting identifiers of all kinds, attempting to make them as short as possible to improve human readability. Finally, we target human-readable compressed identifiers shortening UUID’s as follows;

```
    # 36 character UUID
    00000000-0000-0000-0000-000000000000
    # to 27 characers
    000000.000.000.000.00000000
    # or 22 characters 128/6
    0000000000000000000000
    # or single or short sequences characters for small numbers
    0
```

6.2. Huge Numbers and the 10-6-4 Convention

Ten64 is designed to be used to encode huge numbers, and is also optimized for this use-case. The 10-6-4 convention suggests huge strings of numbers SHOULD be segmented into segments of ten characters, six characters, and then four characters. In addition, these huge strings MAY be separated by the Unix line feed. This also creates a convention of at MOST 92 characters per line SHOULD be used, it is an effort to improve human readability.

```
    #..
    0123456789-abcdef-ghij-k$mnopqrst-uvwyxz-ABCD-EFGH+JKLMN-!PQRST-UVWY-XZ@_012345-67
89ab-cdef
    ghijk$mnop-qrstuv-wyxz-ABCDEFGH+J-KLMN!P-QRST-UVWYXZ@012-345678-9abc-defghijk$m-no
pqrs-tuvw
    yxzABCDEFG-H+JKLM;
    #..
    0123456789-abcdef-ghij-k$mnopqrst-uvwyxz-ABCD-EFGH+JKLMN-!PQRST-UVWY-XZ@_012345-67
89ab-cdef
    ghijk$mnop-qrstuv-wyxz-ABCDEFGH+J-KLMN!P-QRST-UVWY.XZ@012-345678-9abc-defghijk$m-n
opqrs-tuv
    wyxzABCDEF-GH+JKL-M;
```

The above code illustrates a huge integer number and a huge decimal number. These kinds of Ten64 character sequences MAY be included as strings or as template literals in many programming languages. In addition, these huge numbers MAY be streamed from disk or over the network.

7. Binary

Ten64 is also a system to create BitSlotMaps#1.3.6.1.4.1.33097.1.1.3 (aka, BinaryStrings, BitVectors, BitSets, etc). Numbers are read into memory using the Ten64 integer serialization algorithm. Then the binary integer numbers MAY be reinterpreted per the user's wishes.

Ten64 MAY also be used to represent arbitrary octet arrays. Note, conversion to octet arrays is NOT the primary Use-Case of Ten64, and that round tripping between octet arrays MAY introduce issues.

7.1. Ten64 to Octet Array Conversion

When converting Ten64 binary to an array of octets, all missing bits MUST be filled with zeros. This is to ensure that the binary consists of complete octets.

7.2. Ten64 from Octet (Byte) Array Conversion

When converting arrays of octets to the Ten64 alphabet, all '0' characters from the Ten64 alphabet at the right MUST be omitted.

8. Related Technologies

There are a ton of libraries in various languages, for example Java's BigInteger influenced the ECMA Script BigInt. In addition, this ECMA Script BigDecimal implementation is based on Java's BigDecimal. We do NOT expect Ten64 to gain wide adoption over the Modern Western Numeral System, since the Modern Western Numeral System is taught in early elementary schools and used all the way through advanced mathematics classes.

9. Performance

Ten64 significantly reduces the number of characters required for encoding, which saves on disk space in files and on the number of bytes transferred over sockets. Ten64 SHOULD be implemented using a more optimal algorithm to serialize and de-serialize the data than Modern Base-10 Numeral System serialization uses.

To create integers from the Ten64 alphabet, algorithms SHOULD use switch statements to convert the Ten64 alphabet into little-endian binary (used in the majority of in memory number systems). Then the algorithms should shift the bits and use the binary and (i.e. &) operator to aggregate the number into integers. This Ten64 Integer Serialization#1.3.6.1.4.1.33097.0.2.4 Algorithm will complete with a O(s) serialization and O(c) de-serialization time cost. Note; s and c represent the sextets and characters in the previous sentence, respectively.

Comparison with other algorithms which use various serialization and de-serialization forms to and from the Modern Western Numerical System is generally much slower.

* Number Conversion Calculator

* Number Conversion at Instructables

* Number Conversion at Khan Academy

* Number Conversion at Lumen

* Number Conversion at WikiHow

However, when converting decimal numbers using Ten64 Decimal Serialization#1.3.6.1.4.1.33097.0.2.5, division is required. Depending on the division algorithm used, this is roughly comparable to serialization and de-serialization of the Modern Western Numerical System covered above.

10. Compatibility

When drafting Ten64, we went through great lengths to attempt to make it as compatible as possible with the largest number of usage environments. However, it is impossible to have pristine compatibility with such a large variety of usage environments. In short, for use in the REST Programming Style or other URI/HTTP based systems we recommend using Reserved Expansion from 3.2.3 (URI Template Variable Values) with Ten64, ommitting the ’#’ and ’;’.

10.1. URI and URI Template Compatibility

The following Ten64 alphabet characters MAY have issues with URIs and URI Templates. The following section is in order of usage in Ten64.

10.2. Leading Special Characters

10.2.1. URI Pound Symbol '#'

The pound symbol is a protected character by the 3.2 (URIs RFC3986 section 3.2). When using Ten64 inside of URIs (aka URLs), the pound symbol MUST be either omitted or escaped with a percent sign '%23'. Additionally, no major conflicts are foreseen with URI Templates.

10.2.2. URI Minus Symbol '-'

The minus symbol is an unreserved character by the 2.3 (URIs RFC3986 section 2.3). No special treatment of this character is required for URIs. However, 2.3 (URI Template Variable Names) will require encoding as '%2D'.

10.3. Alphabet Characters

10.3.1. URI Dollar Sign Symbol '$'

The dollar sign symbol is a sub-delim character by the 3.4 (URIs RFC3986 section 3.4). It is explicitly permitted in the 3.4 (query component of a URI). Although NOT required by Ten64, some languages MAY require URL encode the dollar symbol. In particular, Bash, PowerShell, PHP, Perl, and Ruby where it is used to trigger variable expansion.

Simple String Expansion in 3.2.2 (URI Template Variable Values) MUST encode the dollar sign '$' as '%24'. However, Reserved Expansion in 3.2.3 (URI Template Variable Values) MUST NOT encode the dollar sign '$'.

10.3.2. URI Plus Symbol '+'

The plus symbol is a sub-delim character by the 3.4 (URIs RFC3986 section 3.4). It is explicitly permitted in the 3.4 (query component of a URI). However, many languages and libraries (PHP, Python's urllib, Java Servlets) automatically decode this as a space, so it MAY need to be escaped as '%2B'.

Simple String Expansion in 3.2.2 (URI Template Variable Values) MUST encode the plus symbol as '%2B'. However, Reserved Expansion in 3.2.3 (URI Template Variable Values) MUST NOT encode the plus symbol '+'.

10.3.3. URI Exclamation Mark '!'

The exclamation mark is a sub-delim character by the 2.2 (URIs RFC3986 section 2.2). It is explicitly permitted in the 3.4 (query component of a URI). However, many languages and libraries (Express.js, Ruby on Rails, and Django) automatically decode this as a space, so it MAY need to be escaped as '%21'.

Simple String Expansion in 3.2.2 (URI Template Variable Values) MUST encode the exclamation mark '!' as '%21'. However, Reserved Expansion in 3.2.3 (URI Template Variable Values) MUST NOT encode the exclamation mark '!'.

10.4. Internal and Trailing Special Characters

10.4.1. URI Period '.'

The period is unreserved in URIs RFC3986. However, it is often used for relative paths, We do NOT anticipate any compatibility issues.

10.4.2. URI Semicolon ';'

The semicolon symbol is a sub-delim character by the 3.4 (URIs RFC3986 section 3.4). It is explicitly permitted in the 3.4 (query component of a URI). There may be legacy issues with Python and Java servlets, which have NOT received security patches. The semicolon MAY be omitted or encoded as '%3B'.

Simple String Expansion in 3.2.2 (URI Template Variable Values) MUST encode the seimcolon symbol ';' as '%3B'. However, Reserved Expansion in 3.2.3 (URI Template Variable Values) MUST NOT encode the semicolon ';'.

10.5. DID Compatibility

We removed the use of the colon ':' and replaced it with an exclamation point '!', specifically to increase compatibility with the DID specification. However depending on usage, Ten64 MAY still require URI encoding for use with DIDs.

```
did:ten64:abc123
```

10.6. DOID Compatibility

DOID uses Ten64 in a downstream manner, so it is fully compatible with Ten64.

10.7. EJCN Compatibility

EJCN (Extensible JSON Classification Notation) uses Ten64 in a downstream manner, so it is fully compatible with Ten64.

10.8. JSON Compatibility

JSON Strings are fully compatible with Ten64, as they use the backslash character '\' for escaping characters.

10.9. Terminal-Shell Compatibility

Various shells (i.e. Bash) will likely have some compatibility issues due to the dollar sign character '$' use when referenceing ENVIRONMENT_VARIABLES and function parameters. This can be overcome by escaping the dollar sign character '$' with a backslash '\$', or by wrapping the text in single quotes.

10.10. XML Compatibility

XML Strings are fully compatible with Ten64, as they use the are distinct from the big five XML characters '<', '>', '&', '"', and '''.

10.11. Other Textual Compatibility

Ten64 SHOULD be generally compatible with most text files and programming syntax. We have intentionally avoided commonly used characters like brackets, braces and parentheses '[',']','{','}','(',')'. In addition, we have avoided common escape characters '%','&', etc.

11. Interpretation Conventions

The Ten64 character sequences may have additional corresponding meta-data information from various schema sources like;

* EJCN (Extensible JSON Classification Notation)
* EJCN (Extensible JSON Classification Notation) Schemas
* JSON Schemas
* XML Schemas

Although the definitions of all these schema types are out of the scope of this document, we supply the following interpretation conventions.

11.1. The Default Interpretation as Integers, Decimals, and Lists of Lists of Integers

The following sequences explode as follows;

```
#0.1;   expands to a list of integers 0, 1 which MAY also be
        interpreted as the decimal 0.1 depending on the context.
#0.1.2; expands to a segmented number / list of integers 0, 1, 2.
#01.11; expands the a list of integers 64, 65 which MAY also be
        interpreted as the decimal 64.65 depending on the context.
#-1.8;  expands to a list of integers -1, 8 and MAY also be
        interpreted as a decimal -1.8 depending on the context.
#-1.-9  expands to a list of integers -1, -9
```

11.2. Segmented Number Interpretations Summary

There are several interpretation Use-Cases for segmented numbers, including Dates, Datetimes, MiliTimestamps, NanoTimestamps, DOIDs, IANA OIDs, Points and more.

11.3. BigDecimal Interpretations

Big Decimals (i.e. Java or Javascript type) can be easily represented with Ten64, which encodes the EXACT number of Decimal Places. This provides an alternative to JSON (decimal) numbers which MAY use IETF Single and Double Precision Floating Point numbers, because of the ECMAScript standard. The other alternative is to encode the Modern Western Numerals as strings, and then use something like a BigDecimal or BigDecimal clone to interpret the text data. See the commentary for a deep dive on this topic.

11.4. Date Interpretations

Dates can be greatly condensed using the Segmented Number base class. Dates should be standardized as the year ten64 followed by a dot, and one ten64 character for the month and one ten64 character for the day. For example;

```
#Vv.21;  expands to 2023-02-01
```

11.5. Datetime Interpretations

Datetimes add the additional timezone, (military time) hour and minute to the date. The timezone, (military time) hour and minute each only take one ten64 character and can be encoded as follows;

```
#Vv.216jR;  expands to 2023-02-01 CST 7:53 PM
```

Note j is 19, in military time -12 = 7 PM.

11.6. List Interpretations

Ten64 supports lists of number separated by commas, each number MAY include whitespace characters on either side of the number;

```
#1,2,3,4;  expands to a number list of 1,2,3,4
#1.2.3,4.5.6,7.8.9;  expands to a list of 3d points
```

11.7. MiliTimestamp Interpretations

MiliTimestamps add a single ten64 character for seconds and separate milliseconds with an additional dot.

```
#Vv.216jR1.2; expands to 2023-02-01 CST 7:53:01.2
or with milliseconds expanded PM 2023-02-01 CST 7:53:01.002 PM
```

11.8. NanoTimestamp Interpretations

NanoTimestamps add an additional dot, as the MiliSeconds can be multiple characters (with values 0-1000), and then multiple ten64 characters representing the additional (0-1,000,000,000) nanoseconds that are NOT tracked as milliseconds.

```
#Vv.216jR1.2.7;  expands to 2023-02-01 CST 7:53:01.2.7 PM
or with nanoseconds expanded 2023-02-01 CST 7:53:01.002000007 PM
```

11.9. Point Interpretations

Ten64 can encode 2d, 3d, and Nd points as segmented numbers.

```
# A 3d decimal point
#-1.7,-5.2,-7.9;
```

11.10. Time Interpretations

Time will use the time segments from the Datetime, MiliTimestpan, and NanoTimestamp.

```
# The Date Time
#Vv.216jR;  expands to 2023-02-01 CST 7:53 PM
# vs just the time part
#6jR;  expands to CST 7:53 PM
```

Note j is 19, in military time -12 = 7 PM.

12. Modern Western Numeral System

Arabic, Ghubari, and Vedic as well as many other numeral systems were considered for use in this RFC. We finally settled down on modern the name *The Modern Western Numeral System*. It appears that numeral systems have influenced each other over the ages, and we will likely continue carbon dating each glyph 0-9 for some time, as we have recently carbon dated the glyph '0'. In addition, even if we do find an older carbon date of a particular glyph, It could take a considerable amount of time to determine if that carbon dating references a different culture than the main culture which recorded the glyph.

* Origin of the Numerals [origin-of-modern-mathematical-numeral]

* Carbon Dating Reveals the History of Zero Is Older Than Previously Thought [carbon-dating-zero]

| Before we go on to analytically review the Hindu-Indian
| Brahmagubta and Islamo-Arabic Ghubari origin of the modern
| mathematical numeral system which is now regarded as the Western
| Numeral System.

* Origin of Modern Mathematical Numeral pg 46 [origin-of-modern-mathematical-numeral]

12.1. Modern Western Integers

Modern Western Integers are simply integers composed using the Modern Western Numerical System. *Modern Western Integers* MAY be positive, negative, or zero.

12.2. Modern Western Decimal Numbers

Modern Western Decimal Numbers are simply numbers using the Modern Western Numeral System, which contain a decimal point.

13. IANA Considerations

Adligo Inc. maintains a Private Enterprise Number (PEN) Object Identifier (OID) registered with IANA. The specific OID allocated for the Ten64 serialization format is 1.3.6.1.4.1.33097.8.1.

Further documentation regarding this registration is available at: https://adligo.github.io/papers.adligo.com/ietf-rfcs/Ten64.html

There are no other IANA considerations for this document.

14. Commentary

This section is non-normative.

To improve human readability, we replaced these characters with their respective characters. The following chart shows the history of this.

21: lower case ’l’ → ’$’
44: upper case ’I’ → ’%’ → ’+’
50: upper case ’O’ → ’?’ → ’:’ → ’!’

2026-03-28 Replaced the % sign with the + sign to make URI (URL) escaping easier. Replaced the question mark with the colon to make URI (URL) escaping easier, and then later on replaced it with the exclamation point to make Ten64 more compatible with DIDs.

Although Ten64 can encode and decode numbers of any size and precision, they are often not human-decipherable. During the creation of this text, there was much discussion about JSON RFCs 4627, 7158, 7159, 8259, JavaScript and ECMA Script Numbers. S Morgan believes that a separate document should be created to address the serialization/de-serialization (aka encoding/decoding) which uses text representing the Modern Western Numeral System specifically. He also suggests something like the following;

12 → int, long or ECMA BigInt, Language specifies.
12.78 → Java BigDecimal style numbers or a new BigNumber type
f12.78 → 32 bit IEEE 754 / Java single floating point decimal numbers
d12.78 → 64 bit IEEE 754 / Java double floating point decimal numbers

Although the current state of the JSON RFC 8259 specification is fairly clear, it has a muddied past, which has created confusion and varying interpretations (i.e. GSON, Jackson and others). This starts with the usage of the term JavaScript in the title, the JS in JSON. It took some time for an actual JavaScript-like specification to emerge as ECMA Script which specifies IEEE 754-2019 Floating Point Numbers. These challenges and issues are not traceable to a single standard, but instead the result of the interaction between at least three standards bodies the IEEE, IETF and ECMA International, and the history of JavaScript and Netscape [2] [3] .

As a side note, the ECMA Script 262 website (https://tc39.es/ecma262) chews up enough resources (processor/RAM I didn’t benchmark it?) that it slows down and crashes browsers on my computer with 64 GB of RAM. However, for the brave people who want to click on these direct links;

* ECMA Script 262 Section 6.1.6 Numeric Types (https://tc39.es/ecma262/#sec-numeric-types)

* ECMA Script 262 Section 6.1.6.1 Language Types Number Type (https://tc39.es/ecma262/#sec-ecmascript-language-types-number-type)

* ECMA Script 262 Section 21 Numbers and Dates (https://tc39.es/ecma262/#sec-numbers-and-dates)

In some ways, these serialization issues appear to be fixed in part by more modern RFC's including the following;

* CBOR RFC 8949 which simply uses a binary format to transfer the floating-point numbers.

* HTTP Structured Fields RFC 9651 which does not target text but HTTP or really UIRs RFC 3986 and puts a tight limitation on decimal numbers, only allowing three decimal digits, which isn't compatible with Bitcoin and other wider decimal number formats.

The culmination of these points result in ubiquitous usage of string wrappers for numbers in tools like JSON, which forces the various parsers to get it right all the time;

```
{ "myJSONDecimalNumber": "12.3" }
```

This essentially defeats the purpose of having a (decimal) number type in JSON.

In addition, printing floating point numbers has been challenging historically. Which has led to the Ry algorithm, which greatly reduced the time cost (asymptotic complexity) of printing floating point numbers in various JVM environments. Ry itself improved on the previous How to Print Floating-Point Numbers Accurately paper by Guy L. Steele Jr. and Jon L White, and was eventually adopted by the JCP. Casual readers are encouraged to watch the Ry video. Then ask themselves the philosophical question: What do you want to see from the following pseudo-code?

```
var f : ieee754Float = f0.3
print(f)
```

Some people would prefer *Option A '0.3'* while others (i.e. S Morgan) would prefer *Option B '0.300048828125'*. S Morgan thinks *Option B* is simpler, likely much faster to print and also provides a more accurate representation of what is actually stored in RAM without any rounding.

In addition, since the Java BigDecimal style isn’t actually a standard from any of these standards bodies, except for maybe the JCP. This means we don’t actually have a solid standard for serializing money (i.e. USD, YEN, BTC, etc). Finally, after reading all of this and the citations to ANSI Math X3.274-1996 - X3.274-1996 AM 1-2000 section 74 in the Java 23 BigDecimal source code, (S Morgan) started to ponder: Is all of this mantissa stuff just too complex?

S Morgan:

From more of a philosophical Category Theory perspective; Are we actually usually doing discrete mathematics and have just added decimal places to help us read the natural numbers?

For example, USD currency serialization and mathematical operations are actually using a natural number of cents. Perhaps we should just run with that and do much of our math with BigIntegers , BigInts, and then reformat the string representation with a decimal point. This would likely give most intuitive users who are just learning programming, math or both for the first time a much lower barrier to entry when performing most elementary to high school math, programming and serialization.

A new *BigNumber* convention could be created on top of this idea leveraging the ISO/IEC 10967 integer datatype section 5.1 / International Math Standard.

```
var b : BigNumber = 12.57
println(b)
println(b.toDiscreteString())
println(b.hasDecimalPoint())
var i : BigNumber = -∞
println(i)
println(i.hasDecimalPoint())
var c = BigNumber = 0.3
println(c)
println(c.toFloat())
// should output the following text
12.57
1257
true
-∞
false
0.3
0.300048828125
```

Then this new *BigNumber* type could potentially be used as the basis for further text-encoding number work with the Modern Western Numeral System. Finally, an open question. What should we do with fractions/repeating numbers (aka. connected overlines) (i.e. 0.0123456789 or 1/7 = 0.142857), another new *BigFraction* type perhaps?

```
# Github Markdown for Connected Overlines
0.0123456789
1/7 = 0.142857
```

15. Citations and Workflow Comments

Finally, note that most of the Github style Markdown to RFC style XML conversion, and citations were generated by Gemini. Also, Gemini Deep Research, found ISO/IEC 10967 / A International Math Standard, and other content that I didn’t track. Finally, note I dictated most of this paper using various voice-to-text AI software, which has given much of it a strangely verbal style. Feel free to reach out if you would like to correct, modify or add anything to this paper.

16. Normative References

[ansi] American National Standards Institute (ANSI), "American National Standards Institute - ANSI Home", <https://www.ansi.org/>.

[ansi-x3274-ibm] IBM Corporation, "Decimal Arithmetic Specification, version 1.70 - Appendix A: The X3.274 subset", Website speleotrove.com/decimal, 7 April 2009, <https://speleotrove.com/decimal/dax3274.html>.

[ansi-x3274-1996] InterNational Committee for Information Technology Standards (INCITS), "Information Technology - Programming Language REXX", ANSI INCITS 274-1996/AMD1-2000 (R2001), 2000, <https://webstore.ansi.org/standards/incits/ansiincits2741996amd12000r2001>.

[ansi-x3274-1996-am-1-2000-section-7.4] InterNational Committee for Information Technology Standards (INCITS), "Information Technology - Programming Language REXX", ANSI INCITS 274-1996/AMD1-2000 (R2001), 2000, <https://webstore.ansi.org/standards/incits/ansiincits2741996amd12000r2001>.

[ASCII-7] American Standards Association, "American Standard Code for Information Interchange (ASCII)", ASA X3.4-1963, 17 June 1963, <https://www.ansi.org/>.

[base58] Sporny, M., "The Base58 Encoding Scheme", Work in Progress, Internet-Draft, draft-msporny-base58-03, 11 February 2024, <https://datatracker.ietf.org/doc/html/draft-msporny-base58-03>.

[base64-rfc-4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[Bitcoin] Nakamoto, S., "Bitcoin: A Peer-to-Peer Electronic Cash System", October 2008, <https://bitcoin.org/bitcoin.pdf>.

[bitslotmaps] Morgan, S., "BitSlotMaps", Adligo Papers 1.3.6.1.4.1.33097.1.1.3, 25 November 2025, <https://adligo.github.io/papers.adligo.com/data_structures/BitSlotMaps.html>.

[carbon-dating-zero] Katz, B., "Carbon Dating Reveals the History of Zero Is Older Than Previously Thought", Smithsonian Magazine Smart News, 14 September 2017, <https://www.smithsonianmag.com/smart-news/dating-ancient-indian-text-gives-new-timeline-history-zero-180964896/>.

[category-theory-b-milewski-youtube] Milewski, B., "Category Theory 1.1: Motivation and Philosophy", Video YouTube, 25 August 2016, <https://www.youtube.com/watch?v=I8LbkfSSR58>.

[CBOR-RFC-8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 8949, STD 94, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[decentralized-identifiers-dids] Sporny, M., Guy, A., Sabadello, M., Reed, D., and M. Sporny, "Decentralized Identifiers (DIDs) v1.0", W3C Recommendation REC-did-core-20220719, 19 July 2022, <https://www.w3.org/TR/2022/REC-did-core-20220719/>.

[discrete-mathematics-o-levin-2024]
Levin, O., "Discrete Mathematics: An Open Introduction, 4th Edition", Format PDF, 2024, <https://discrete.openmathbooks.org/pdfs/dmoi4.pdf>.

[doid-repo]
Adligo, "doid.adligo.org: Domain Oracle Identifiers", GitHub Repository, October 2024, <https://github.com/adligo/doid.adligo.org>.

[ejcn-extensible-json-classification-notation]
Adligo, "ejcn.adligo.org: Extensible JSON Classification Notation", GitHub Repository, March 2026, <https://github.com/adligo/ejcn.adligo.org>.

[ejcn-extensible-json-classification-notation-schemas]
Adligo, "EJCN (Extensible JSON Classification Notation) Schemas", GitHub Repository, 2026, <https://github.com/adligo/ejcn_schemas.adligo.org>.

[endian-wikipedia]
Wikipedia contributors, "Endianness", Encyclopedia Wikipedia, The Free Encyclopedia, URL https://en.wikipedia.org/wiki/Endianness, March 2024, <https://en.wikipedia.org/wiki/Endianness>.

[floating-point]
IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2019, DOI 10.1109/IEEESTD.2019.8766229, 22 July 2019, <https://ieeexplore.ieee.org/document/8766229>.

[Floating-Point-Gordon]
Gordon College, "The IEEE 754 Floating-Point Standard", MAT342 Numerical Analysis Course Material, 2024, <https://cs.gordon.edu/courses/mat342/handouts/ieee.html>.

[Floating-Point-J-Burkardt]
Burkardt, J., "IEEE Floating Point Numbers", Department of Scientific Computing Resource, 2023, <https://people.sc.fsu.edu/~jburkardt/html/ieee.html>.

[Floating-Point-Printing]
Shewchuk, J. R., "Adaptive Floating-Point Summation and Arbitrary Precision Floating-Point Arithmetic", DOI 10.1145/1806596.1806623, October 1997, <https://dl.acm.org/doi/10.1145/1806596.1806623>.

[Floating-Point-Wikipedia]
Wikipedia, "IEEE 754: Standard for Floating-Point Arithmetic", Wikipedia, The Free Encyclopedia, October 2023, <https://en.wikipedia.org/wiki/IEEE_754>.

[Grisu3] Loitsch, F., "Printing floating-point numbers quickly and accurately with integers", ACM SIGPLAN Notices vol. 45, no. 6, pp. 233-243, June 2010, <https://doi.org/10.1145/1806651.1806623>.

[HTTP-Structured-Fields-RFC-9651]
Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <https://www.rfc-editor.org/info/rfc9651>.

[hexadecimal]
Wikipedia Contributors, "Hexadecimal", Wikipedia Hexadecimal, March 2026, <https://en.wikipedia.org/wiki/Hexadecimal>.

[iana-oids]
IANA, "Private Enterprise Numbers (PEN)", IANA Registry, March 2026, <https://www.iana.org/assignments/enterprise-numbers>.

[ieee] IEEE, "Institute of Electrical and Electronics Engineers (IEEE)", <https://www.ieee.org>.

[IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2019, July 2019, <https://ieeexplore.ieee.org/document/8766229>.

[IETF] IETF, "Internet Engineering Task Force (IETF)", <https://www.ietf.org>.

[Information-Theory-Elements-Cover-Thomas-2006]
Cover, T. M. and J. A. Thomas, "Elements of Information Theory", DOI 10.1002/047174882X, Publisher Wiley-Interscience, Edition 2nd, July 2006, <https://doi.org/10.1002/047174882X>.

[JavaScript-Wikipedia]
Wikipedia Contributors, "JavaScript", Wikipedia, The Free Encyclopedia Online, 29 March 2026, <https://en.wikipedia.org/wiki/JavaScript>.

[jcp] Java Community Process (Oracle Corporation), "The Java Community Process(SM) Program - Home", <https://www.jcp.org/en/home/index>.

[JSON-RFC-4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <https://www.rfc-editor.org/info/rfc4627>.

[JSON-RFC-7158] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7158, DOI 10.17487/RFC7158, March 2013, <https://www.rfc-editor.org/info/rfc7158>.

[JSON-RFC-7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[json-rfc-8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, STD 90, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[json-schemas] Andrews, H. and A. Wright, "JSON Schema: A Media Type for Describing JSON Data Structures", December 2020, <https://json-schema.org/draft/2020-12/json-schema-core.html>.

[MathAsympoticProcessorPerformanceWikipedia] Wikipedia, "Computational complexity of mathematical operations", Wikipedia, The Free Encyclopedia, October 2023, <https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations>.

[ISO-IEC-10967-1] International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC), "Information technology -- Language independent arithmetic -- Part 1: Integer and floating point arithmetic", ISO/IEC 10967-1:2012, 15 July 2012, <https://www.iso.org/standard/51317.html>.

[Mathematical-Theory-of-Communication-Shannon-1948]
Shannon, C. E., "A Mathematical Theory of Communication", Bell System Technical Journal Vol. 27, pp. 379423, 623656, DOI 10.1002/j.1538-7305.1948.tb01338.x, July 1948, <https://doi.org/10.1002/j.1538-7305.1948.tb01338.x>.

[natural-numbers-wikipedia]
Wikipedia contributors, "Natural number", Website Wikipedia, The Free Encyclopedia, <https://en.wikipedia.org/wiki/Natural_number>.

[network-order-ibm]
IBM, "Network byte order and host byte order", <https://www.ibm.com/docs/ja/zvm/7.2.0?topic=domains-network-byte-order-host-byte-order>.

[origin-of-modern-mathematical-numeral]
Musa, A., "Origin of Modern Mathematical Numeral 0, 1, 2, 3, 4, 5, 6, 7, 8, 9: the Hindu-Indian-Brahmagubta, The Islamo-Arabic or the West?", Page 46, 2014, <https://www.academia.edu/9099310/Origin_of_Modern_Mathematical_Numeral_0_1_2_3_4_5_6_7_8_9_the_Hindu_Indian_Brahmagubta_The_Islamo_Arabic_or_the_West>.

[netscape] Wilson, B., "Browser History: Netscape", Website Index DOT Html/Css (BlooBerry), <http://www.blooberry.com/indexdot/history/netscape.htm>.

[netscape-2]
Pignol, G., "Netscape’s Rise and Fall: A Browser Wars History", Website Medium, 3 December 2025, <https://medium.com/@gp2030/netscapes-rise-and-fall-a-browser-wars-history-8546e3b52092>.

[netscape-3]
Wikipedia contributors, "Netscape", Website Wikipedia, The Free Encyclopedia, <https://en.wikipedia.org/wiki/Netscape>.

[number-conversion-calculator]
RapidTables, "Decimal to Binary Converter: 756", 2026, <https://www.rapidtables.com/convert/number/decimal-to-binary.html?x=756>.

[Number-Conversion-Instructables]
Instructables, "How to Convert From Decimal to Binary", Instructables Science and Tech Category, October 2023, <https://www.instructables.com/How-to-Convert-From-Decimal-to-Binary/>.

[Number-Conversion-Khan-Academy]
Khan, S., "Large Number Decimal to Binary", Video Tutorial: Algebra / Alternate Number Bases, 2026, <https://www.khanacademy.org/math/algebra-home/alg-intro-to-algebra/algebra-alternate-number-bases/v/large-number-decimal-to-binary>.

[Number-Conversion-Lumen]
Lumen Learning, "Converting Between Bases", Waymaker Mathematics for Liberal Arts, 2024, <https://courses.lumenlearning.com/waymakermath4libarts/chapter/converting-between-bases/>.

[Number-Conversion-WikiHow]
WikiHow, "How to Convert from Decimal to Binary", WikiHow Technology Category, 14 September 2023, <https://www.wikihow.com/Convert-from-Decimal-to-Binary>.

[OData-Protocol]
Pizzo, M., Handl, R., and M. Zurmuehl, "OData Version 4.01. Part 1: Protocol", OASIS Standard OData-v4.01-Part1, June 2021, <https://docs.oasis-open.org/odata/odata/v4.01/os/part1-protocol/odata-v4.01-os-part1-protocol.html>.

[Radix-10-vs-60-Research-Gate]
ResearchGate Community, "Why we use base-10 almost everywhere than base-60 which was first invented method?", URL https://www.researchgate.net/post/Why-we-use-base-10-almost-everywhere-than-base-60-which-was-first-invented-method, May 2014, <https://www.researchgate.net/post/Why-we-use-base-10-almost-everywhere-than-base-60-which-was-first-invented-method>.

[rest-programming-style]
Fielding, R. T., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation University of California, Irvine, 2000, <https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf>.

[radix-wikipedia]
Wikipedia contributors, "Radix", Encyclopedia Wikipedia, The Free Encyclopedia, URL https://en.wikipedia.org/wiki/Radix, March 2024, <https://en.wikipedia.org/wiki/Radix>.

[RFC20] Cerf, V., "ASCII format for Network Interchange", RFC 20, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996, <https://www.rfc-editor.org/info/rfc2026>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119", BCP 14, RFC 8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., "The JSON Data Interchange Format", STD 90, RFC 8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[ry] Adams, U., "Ry: fast float-to-string conversion", Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation PLDI 2018, pp. 270-282, June 2018, <https://doi.org/10.1145/3192366.3192369>.

[ry-video]
Adams, U., "Ry: Fast Float-to-String Conversion (PLDI 2018)", Video YouTube, 2018, <https://www.youtube.com/watch?v=kw-U6smcLzk&t=457s>.

[octet] Wikipedia Contributors, "Octet", Wikipedia Octet, March 2026, <https://en.wikipedia.org/wiki/Octet_(computing)>.

[positional-number-systems-wikipedia]
Wikipedia contributors, "Numeral system: Positional systems in detail", Encyclopedia Wikipedia, The Free Encyclopedia, March 2024, <https://en.wikipedia.org/wiki/Numeral_system#Positional_systems_in_detail>.

[sextet] Wikipedia Contributors, "Sextet", Wikipedia Sextet, March 2026, <https://en.wikipedia.org/wiki/Units_of_information#sextet>.

[ten64-decimal-serialization-algorithm] Adligo, "Ten64DecimalSerialization: A Concrete Algorithm for 64-bit Decimal Representation", Adligo Algorithm Specification Series, 2024, <https://adligo.github.io/papers.adligo.com/algorithms/concrete/Ten64DecimalSerialization.html>.

[ten64-integer-serialization-algorithm] Adligo, "Ten64IntegerSerialization: A Concrete Algorithm for 64-bit Integer Representation", Adligo Algorithm Specification Series, 2024, <https://adligo.github.io/papers.adligo.com/algorithms/concrete/Ten64IntegerSerialization.html>.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, STD 63, November 2003, <https://www.rfc-editor.org/rfc/rfc3629>.

[uri-rfc-3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[URI-Templates-RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[uuid-rfc-9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

[xml-schemas] Peterson, D., Gao, S., Malhotra, A., Sperberg-McQueen, C., and H. Thompson, "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes", W3C Recommendation REC-xmlschema11-2-20120405, 5 April 2012, <https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/>.

17. Informative References

[BigInt-Java]
Oracle Corporation, "Java Platform, Standard Edition v21: Class BigInteger", 2023, <https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/math/BigInteger.html>.

[BigDecimal-Java]
Oracle Corporation, "Java Platform, Standard Edition v21: Class BigDecimal", Java Standard Library, 2023, <https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/math/BigDecimal.html>.

[BigDecimal-NPM]
STZ-IDA, "BigDecimal: Arbitrary-precision decimal arithmetic", NPM Package, September 2024, <https://www.npmjs.com/package/bigdecimal>.

[bioctal] Community, "Bioctal: Hexadecimal 2.0", <https://en.wikipedia.org/wiki/Bioctal>.

[OriginOfTheNumerals]
Boucenna, A., "Origin of the numerals", arXiv math/0606699, 30 June 2006, <https://arxiv.org/abs/math/0606699>.

[CACM-Binary]
ACM, "Letters to the editor: On binary notation", DOI 10.1145/364096.364107, 1968, <https://doi.org/10.1145/364096.364107>.

[ECMA-262] Ecma International, "ECMAScript 2025 Language Specification", Standard ECMA-262, June 2025, <https://tc39.es/ecma262/>.

[google-gemini]
Emergent Mind, "Google Gemini: Scalable Multimodal Models", Website Emergent Mind, 14 January 2026, <https://www.emergentmind.com/topics/google-gemini>.

[google-gemini-deep-research]
i10X, "Google Gemini Deep Research: AI for Complex Tasks", Website i10X, 16 December 2025, <https://i10x.ai/news/google-gemini-deep-research-analysis>.

[Gson-GitHub]
Google, "Gson: A Java serialization/deserialization library to convert Java Objects into JSON and back", GitHub repository, March 2026, <https://github.com/google/gson>.

[Jackson-GitHub]
FasterXML, LLC, "Jackson: Main Portal page for the Jackson project", GitHub repository, March 2026, <https://github.com/fasterxml/jackson>.

[IETF-Style]
Internet Engineering Task Force (IETF), "Language and Style Guide for IETF Authors", IETF Author Resources, 2024, <https://authors.ietf.org/language-and-style>.

[ISO8601] ISO, "Representation of dates and times", ISO 8601, 2004, <https://www.iso.org/standard/40874.html>.

[Smithsonian-Zero]
Katz, B., "Carbon Dating Reveals the History of Zero Is Older Than Previously Thought", Smart News, 14 September 2017, <https://www.smithsonianmag.com/smart-news/dating-ancient-indian-text-gives-new-timeline-history-zero-180964896/>.

[W3C.REC-xml-20081126]
Bray, T., "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C REC REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>.

Acknowledgments

The author wishes to acknowledge R Ismo, who was instrumental in providing meticulous scrutiny to this project. Although he disagreed and MAY still disagree with much of it, the discourse was wonderful and fully worthwhile.

Author’s Address

Scott Morgan
Adligo Inc.
Email: scott@adligo.com
URI: https://github.com/adligo/ten64.adligo.org