Network Working Group
Internet-Draft
Intended status: Informational
Expires: 12 February 2026

Rayner
Independent

11 August 2025

Proquints: Readable, Spellable, and Pronounceable Identifiers
draft-rayner-proquint-03

Abstract

This document specifies "proquints" (PRO-nounceable QUINT-uplets), a human-friendly encoding that maps binary data to pronounceable identifiers using fixed consonant-vowel patterns. The concept was originally described by Daniel Shawcross Wilkerson in 2009. This document formalizes the format for archival and reference.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 February 2026.

Copyright Notice

Table of Contents

1. Introduction

Proquints encode binary data as alternating consonant-vowel letters grouped into five-letter syllables, yielding identifiers that are readable, spellable, and pronounceable. The idea and specific letter tables were first described by Daniel Shawcross Wilkerson in 2009 ([WILKERSON2009]). This document does not claim originality for the concept; it reformulates and formalizes the description for archival purposes.

While multiple schemes exist for encoding network addresses and other binary data, Proquints aim to provide a unique blend of human-reabability, accessibility, and long-term usability. They reduce transcription errors, are friendlier for non-technical users, and offer mnemonic qualities that can help in educational or operational contexts. Although they may not replace all existing representations, Proquints can serve as a complementary format that improves clarity in documentation, user interfaces, and spoken communication, particularly where accuracy and inclusivity matter.

2. 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.

3. Format

A proquint encodes data in 16-bit blocks. Each block maps to a five-letter syllable of the form CVCVC (Consonant-Vowel-Consonant-Vowel-Consonant).

The mapping tables are fixed:

Consonants (indices 0..15):

* b d f g h j k l m n p r s t v z

Vowels (indices 0..3):

* a i o u

4. Encoding

* Split the input byte string into 16-bit words (big-endian). If the number of bytes is odd, an implementation MAY pad a single zero byte to complete the final word; if padding is used, applications MUST define how the original length is recovered.

* For each 16-bit word, map bits 15-12 to the first consonant, bits 11-10 to the first vowel, bits 9-6 to the second consonant, bits 5-4 to the second vowel, and bits 3-0 to the final consonant.

* Concatenate syllables. Hyphens MAY be inserted between syllables for readability; decoders MUST ignore hyphens.

5. Decoding

Decoders MUST reverse the mapping in Section 4. Each five-letter syllable maps to one 16-bit value using the same tables and bit ordering. Hyphens, if present, MUST be ignored.

6. Encoding and Decoding Specification

6.1. Letter Tables and Indices

Proquint encodes each 16-bit word as five letters in the pattern CVCVC (Consonant寰天owel寰鼎onsonant寰天owel寰鼎onsonant). The mapping tables and indices are fixed and normative.

Consonant table (index 0..15):

Index	Hex	Bits	Consonant
0	0	0000	b
1	1	0001	d
2	2	0010	f
3	3	0011	g
4	4	0100	h
5	5	0101	j
6	6	0110	k
7	7	0111	l
8	8	1000	m
9	9	1001	n
10	A	1010	p
11	B	1011	r
12	C	1100	s
13	D	1101	t
14	E	1110	v
15	F	1111	z

Vowel table (index 0..3):

Index	Bits	Vowel
0	00	a
1	01	i
2	10	o
3	11	u

6.2. Bit Layout

Each 16-bit input value (bits 15..0, most significant bit first) MUST be mapped to letters in this order:

```
bits 15..12 -> first consonant (C1)
bits 11..10 -> first vowel     (V1)
bits  9.. 6 -> second consonant(C2)
bits  5.. 4 -> second vowel    (V2)
bits  3.. 0 -> third consonant (C3)
```

Encoders MUST process input as an ordered sequence of 16-bit words formed from the input octet string in network byte order (big-endian): octet[i] contributes bits 15..8 and octet[i+1] contributes bits 7..0 of the word. If the input contains an odd number of octets, encoders MAY pad a single zero octet to complete the final 16-bit word; applications using padding MUST specify how the original length is recovered.

Encoders MAY insert ASCII hyphens (0x2D) between syllables for readability. Decoders MUST ignore hyphens.

6.3. Encoding Algorithm (Pseudocode)

```
Input: bytes[]  // octet string
Output: string  // proquint

consonants = "bdfghjklmnprstvz"   // length 16, index 0..15
vowels     = "aiou"               // length 4,  index 0..3

function encode(bytes):
  out = ""
  i = 0
  while i < len(bytes):
    hi = bytes[i]; i += 1
    if i < len(bytes): lo = bytes[i]; i += 1
    else:              lo = 0  // optional pad
    w = (hi << 8) | lo        // 16-bit word

    c1 = consonants[(w >> 12) & 0xF]
    v1 = vowels    [(w >> 10) & 0x3]
    c2 = consonants[(w >>  6) & 0xF]
    v2 = vowels    [(w >>  4) & 0x3]
    c3 = consonants[(w      ) & 0xF]

    syllable = c1 + v1 + c2 + v2 + c3
    out += syllable
    // optional readability: insert '-' between syllables
    // e.g., if not last: out += '-'

  return out
```

6.4. Decoding Algorithm (Pseudocode)

```
Input: string pq  // CVCVC syllables, hyphens optional
Output: bytes[]   // octet string

consonants = "bdfghjklmnprstvz"
vowels     = "aiou"

function indexOf(ch, table):
  pos = table.find(ch)
  if pos < 0: error("invalid character")
  return pos

function decode(pq):
  // remove hyphens; decoders MUST accept upper or lower case
  s = toLowercase(removeAll(pq, '-'))
  if len(s) % 5 != 0: error("length not multiple of 5")

  out = []
  for j in range(0, len(s), 5):
    c1 = indexOf(s[j+0], consonants)
    v1 = indexOf(s[j+1], vowels)
    c2 = indexOf(s[j+2], consonants)
    v2 = indexOf(s[j+3], vowels)
    c3 = indexOf(s[j+4], consonants)

    w = (c1 << 12) | (v1 << 10) | (c2 << 6) | (v2 << 4) | c3
    out.append( (w >> 8) & 0xFF )
    out.append(  w       & 0xFF )

  return out
```

Decoders MUST accept input in either case (upper/lower) and MUST reject any character not in the defined consonant/vowel sets (after stripping hyphens). If applications use padding on encode, they MUST specify how to remove any trailing zero octet introduced solely for padding.

6.5. Normalization

Encoders SHOULD produce lowercase output. Decoders MUST treat input as case-insensitive and MUST ignore ASCII hyphens (0x2D).

Encoders and decoders MUST use the tables and ordering defined in Section 6.1 and Section 6.2. Substituting letters or re-ordering bits is not Proquint and will not interoperate.

6.6. Test Vectors

The following vectors are derived directly from this specification and can be used to verify independent implementations.

```
# Single-word (16-bit) values:
0x0000 -> babab
0xFFFF -> zvzuz
0x1234 -> damuh
0xF00D -> zabat
0xBEEF -> ruroz

# Two words (32-bit), big-endian byte order:
bytes:  0x12 0x34 0xF0 0x0D
words:  0x1234, 0xF00D
pq:     damuh-zabat      (with hyphen)  or  damuhzabat (without)

# Raw ASCII example ("F3r41OutL4w"),
# UTF-8 bytes, zero-padded to even length:
ASCII:  46 33 72 34 31 4F 75 74 4C 34 77
Pad:                                    00
Words:  0x4633 0x7234 0x314F 0x7574 0x4C34 0x7700
PQ:     himug-lamud-kudaz-lijuh-hubuh-lisab  (hyphens optional)
```

Implementations MUST reproduce these outputs exactly.

6.7. Error Handling

Decoders MUST fail input that: (1) contains characters outside the defined tables (after hyphen removal); (2) has length not divisible by 5 letters; or (3) violates the CVCVC pattern. Error signaling is application-specific but MUST reject invalid input rather than attempt to guess.

7. Security Considerations

Proquint is a presentation encoding. It provides no confidentiality, integrity, or authentication services. It does not add or remove entropy, and it MUST NOT be used as a cryptographic transform.

8. IANA Considerations

This document has no IANA actions.

9. Acknowledgments

The author thanks Daniel Shawcross Wilkerson for originating the proquint concept and publishing the initial specification in 2009 ([WILKERSON2009]).

10. References

10.1. Normative References

[BCP14] Best Current Practice 14, <https://www.rfc-editor.org/info/bcp14>. At the time of writing, this BCP comprises the following:

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

10.2. Informative References

[WILKERSON2009] Wilkerson, D.S., "Proquints: Identifiers that are Readable, Spellable, and Pronounceable", arXiv 0901.4016, January 2009, <https://arxiv.org/html/0901.4016>.

Author’s Address

Thomas Rayner
Independent
Email: thmsrynr@outlook.com