

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 21 February 2026

Rayner
Independent
20 August 2025

Proquints: Readable, Spellable, and Pronounceable Identifiers
draft-rayner-proquint-05

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 21 February 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction	2
2. Requirements Language	3
3. Format	3
4. Encoding	3
5. Decoding	4
6. Encoding and Decoding Specification	4
6.1. Letter Tables and Indices	4
6.2. Bit Layout	5
6.3. Encoding Algorithm (Pseudocode)	5
6.4. Decoding Algorithm (Pseudocode)	6
6.5. Normalization	8
6.6. Test Vectors	8
6.7. Error Handling	10
6.8. Backward Compatibility	10
7. Security Considerations	10
8. IANA Considerations	10
9. Acknowledgments	10
10. References	10
10.1. Normative References	10
10.2. Informative References	11
Author's Address	11

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-reability, 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

- * Encoders MUST process the input as an ordered sequence of 16-bit words formed from the octet string in network byte order (big-endian).
- * If the input contains an odd number of octets, encoders MUST append a single zero octet (0x00) to complete the final 16-bit word and MUST signal this padding by appending a single trailing hyphen (U+002D HYPHEN-MINUS) to the end of the proquint string. Encoders MUST NOT append a trailing hyphen when the input length is even.
- * Hyphens between syllables remain optional for readability; decoders MUST ignore interior hyphens. Only a single trailing hyphen has special meaning as a padding signal; multiple trailing hyphens are invalid.
- * 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 interior hyphens.

5. Decoding

- * Decoders MUST reverse the mapping in Section 4.
- * Decoders MUST accept upper- or lower-case input and MUST ignore interior hyphens. If the input ends in a single trailing hyphen, the decoder MUST: (1) decode the syllables to octets; (2) verify that the final octet is 0x00; and (3) remove that final octet. If a trailing hyphen is present and the final octet is not 0x00, the decoder MUST treat the input as invalid.
- * If no trailing hyphen is present, the decoder MUST NOT remove any trailing octet, even if it is 0x00.
- * Inputs with multiple trailing hyphens, a trailing hyphen without any syllables, or a length not divisible by five letters (after removing hyphens) MUST be rejected.

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 ㄆㄣ ㄣ ㄣ ㄣ ㄣ onsonant ㄆㄣ ㄣ ㄣ ㄣ ㄣ). 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 interior hyphens, but not trailing hyphens which indicate padding.

6.3. Encoding Algorithm (Pseudocode)

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

consonants = "bdfghjklmnpqrstvz"
vowels     = "aiou"

function encode(bytes):
    out = ""
    i = 0
    pad = false

    while i < len(bytes):
        hi = bytes[i]; i += 1
        if i < len(bytes):
            lo = bytes[i]; i += 1
        else:
            lo = 0x00
            pad = true

        w = (hi << 8) | lo
        c1 = consonants[(w >> 12) & 0xF]
        v1 = vowels     [(w >> 10) & 0x3]
        c2 = consonants[(w >>  6) & 0xF]
        v2 = vowels     [(w >>  4) & 0x3]
        c3 = consonants[(w          ) & 0xF]
        out += c1 + v1 + c2 + v2 + c3
        // optional: insert interior '-' between syllables for readability

    if pad and len(out) > 0:
        out += '-' // trailing hyphen signals padding was added

    return out
```

6.4. Decoding Algorithm (Pseudocode)

```
Input: string pq // CVCVC syllables, hyphens optional
           // (except final hyphen is meaningful)
Output: bytes[] // octet string

consonants = "bdfghjklmnpqrstvz"
vowels     = "aiou"

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

function decode(pq):
```

```
pad = false
if len(pq) > 0 and pq[-1] == '-':
    pad = true
    if len(pq) >= 2 and pq[-2] == '-':
        error("multiple trailing hyphens")
    pq = pq[0:-1]

if len(pq) > 0 and pq[0] == '-':
    error("leading hyphen not allowed")
if "---" in pq:
    error("consecutive interior hyphens not allowed")

s = toLowercase(pq)
parts = (s == "" ? [] : split(s, '-'))

if len(parts) == 0:
    error("empty input not allowed")
for part in parts:
    if len(part) != 5:
        error("syllable length must be 5")
    c1,v1,c2,v2,c3 = part[0],part[1],part[2],part[3],part[4]
    if indexOf(c1, consonants) < 0
        or indexOf(c2, consonants) < 0
        or indexOf(c3, consonants) < 0:
        error("invalid consonant")
    if indexOf(v1, vowels) < 0 or indexOf(v2, vowels) < 0:
        error("invalid vowel")

out = []
for part in parts:
    c1,v1,c2,v2,c3 = part[0],part[1],part[2],part[3],part[4]
    w = (indexOf(c1,consonants) << 12) |
        (indexOf(v1,vowels) << 10) |
        (indexOf(c2,consonants) << 6) |
        (indexOf(v2,vowels) << 4) |
        (indexOf(c3,consonants) )
    out.append((w >> 8) & 0xFF)
    out.append( w & 0xFF)

if pad:
    if len(out) == 0 or out[-1] != 0x00:
        error("trailing hyphen requires a final 0x00 padding byte")
    out.pop() // remove the padding byte

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. Encoders MUST append a single trailing hyphen only when signaling padding (odd input length). Decoders MUST treat input as case-insensitive, MUST ignore interior hyphens, and MUST apply the trailing-hyphen padding rule defined in this document.

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 -> zuzuz
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 ("F3r4lOutL4w"),

UTF-8 bytes, zero-padded to even length:

ASCII: 46 33 72 34 31 4F 75 74 4C 34 77

Length: 11 bytes

Pad: 00

Words: 0x4633 0x7234 0x314F 0x7574 0x4C34 0x7700

PQ: himug-lamuh-gajaz-lijuh-hubuh-lisab- (interior hyphens optional)

Padding examples

Even-length input (no padding, no trailing hyphen):

bytes: 01 02 03 00

words: 0x0102, 0x0300

pq: bahaf-basab (or "bahafbasab" without interior hyphen)

out: 01 02 03 00

Odd-length input with padding signaled by trailing hyphen:

bytes: 01 02 03

encoder pads: -> add 00 to form final word 0x0300

pq: bahaf-basab- (trailing hyphen REQUIRED)

decoder: decodes to 01 02 03 00, verifies last octet 00, then removes it

out: 01 02 03

Invalid (trailing hyphen but last octet != 00):

pq: bahaf-basad-

-> decode last word to ... 01 (not 00) => ERROR

Invalid (multiple trailing hyphens):

pq: bahaf-basab-- => ERROR

Implementations MUST reproduce these outputs exactly.

6.7. Error Handling

Decoders MUST fail input that: (1) contains characters outside the defined tables (after interior 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.

A trailing hyphen MUST only be used to signal removal of a single trailing 0x00 octet; any other usage is invalid.

6.8. Backward Compatibility

Implementations that predate this specification 遗留 padding specification may ignore a trailing hyphen and therefore retain the trailing 0x00 octet. To interoperate with such decoders, producers SHOULD avoid relying on padding removal when communicating with unknown peers.

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.

Use of a trailing hyphen reveals the parity of the original octet length. This leaks at most one bit of information (even vs. odd length), which is unlikely to be security-relevant for typical uses of Proquints.

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