Network Working Group
Internet-Draft
Intended status: Informational
Expires: 16 March 2026

Rayner
Independent

12 September 2025

# Proquints: Readable, Spellable, and Pronounceable Identifiers
draft-rayner-proquint-08

## 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 16 March 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. Applicability to Networking

While Proquints are general-purpose, they address concrete needs in networked systems and operations. They provide a reversible, human-friendly representation for binary identifiers commonly encountered by implementers and operators, including:

* IP addresses and prefixes (e.g., IPv4 32-bit values map to two syllables; IPv6 128-bit values map to eight syllables).

* Transport and application port numbers (16-bit) and other protocol fields carried in diagnostics.

* Node, interface, device, and request identifiers in distributed systems, where verbal or low-bandwidth exchange occurs during incident response or field operations.

* Log correlation and ticketing, where minimizing transcription errors improves mean time to resolution.

Proquints use only letters [a窶ｦz] and the ASCII hyphen (U+002D). The resulting tokens are case-insensitive and label-safe for many existing systems (e.g., DNS label contents, filenames, and URLs), subject to each system窶ｦ length limits. For example, a 32-bit IPv4 address encodes into two syllables (10 letters), and a 16-bit port number into one syllable (5 letters).

Proquints are intended to complement, not replace, existing textual forms (e.g., dotted-decimal IPv4, IPv6). Operators MAY use Proquints in user interfaces, logs, documentation, and voice communication when human factors (readability, memorability, error-resistance) are advantageous.

*Process note:* This document is submitted to the Independent Stream to provide a stable archival reference for implementers and operators. If substantial community interest develops in standardizing protocol use of Proquints, the work MAY later be dispatched to the IETF for further processing.

## 3. 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.

## 4. 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

## 5. 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. Inserting hyphens between syllables is encouraged to enhance human readability, despite not being mandatory.

## 6. Decoding

* Decoders MUST reverse the mapping in Section 5.

* 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.

## 7. Encoding and Decoding Specification

### 7.1. 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 |

### 7.2. 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.

### 7.3. Encoding Algorithm (Pseudocode)

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

  consonants = "bdfghjklmnprstvz"
  vowels     = "aiou"

  function encode(bytes):
    if len(bytes) == 0: error("empty input not allowed")

    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
```

### 7.4. Decoding Algorithm (Pseudocode)

```
   Input: string pq  // CVCVC syllables; interior hyphens optional;
                     // final hyphen signals padding
   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):
     pq = toLowercase(pq)

     pad = false
     if length(pq) > 0 and pq[-1] == '-':
       pad = true
       if length(pq) >= 2 and pq[-2] == '-':
         error("multiple trailing hyphens")
       pq = pq[0:-1]   // remove the single trailing '-'

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

     if pq == "": error("empty input not allowed")

     // If hyphens present:
     //   split on '-' (no empty chunks allowed)
     // If no hyphens:
     //   input MUST be non-empty and a multiple of 5,
     //   then slice every 5 chars
     parts = []
     if contains(pq, "-"):
       parts = split(pq, "-")
       if any(p == "" for p in parts): error("invalid empty syllable")
     else:
       if (length(pq) % 5) != 0:
         error("run-on form length must be a multiple of 5")
       for i in range(0, length(pq), 5):
         parts.append(pq[i:i+5])

     out = new bytes[2 * length(parts)]
     k = 0
     for part in parts:
       if length(part) != 5: error("syllable length must be 5")
       c1 = indexOf(part[0], consonants)
       v1 = indexOf(part[1], vowels)
       c2 = indexOf(part[2], consonants)
       v2 = indexOf(part[3], vowels)
       c3 = indexOf(part[4], consonants)
       w = (c1 << 12) | (v1 << 10) | (c2 << 6) | (v2 << 4) | c3
       out[k]   = (w >> 8) & 0xFF
       out[k+1] =  w       & 0xFF
       k += 2

     if pad:
       if k == 0 or out[k-1] != 0x00:
```

```
        error("trailing hyphen requires final 0x00 padding byte")
      k -= 1   // drop the padding byte

    return out[0:k]
```

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.

### 7.5. 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 7.1 and Section 7.2. Substituting letters or re-ordering bits is not Proquint and will not interoperate.

### 7.6. 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 ("F3r41OutL4w"),
# 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.

### 7.7. 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.

### 7.8. Backward Compatibility

Implementations that predate this specification寰冭 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.

## 8. Security Considerations

This document defines a reversible textual encoding. It provides no confidentiality, integrity, or authenticity by itself.

*Threat: Loss of confidentiality.* Proquints are a lossless, human-friendly representation of binary data. Encoding sensitive identifiers (e.g., keys, tokens, internal IDs) as Proquints does not hide their value and may make them easier to read, speak, or copy.

_Remediation:_ Treat Proquints with the same confidentiality as the underlying data. When transporting sensitive information, use authenticated encryption or protected channels appropriate to the application (e.g., TLS, SSH, end-to-end encryption). Avoid placing sensitive Proquints in unauthenticated logs, screenshots, or voice channels.

*Threat: Misuse as secrets or passwords.* Because Proquints are pronounceable, implementers might be tempted to use them directly as passwords or shared secrets.

_Remediation:_ Proquints MUST NOT be used as standalone authentication secrets unless generated with appropriate entropy and policy for that purpose. Where human entry is required, rely on established secret-generation and storage practices; do not assume pronounceability confers security.

*Threat: Transcription and spoofing errors.* Human reproduction (reading, hearing, typing) can introduce errors or social-engineering opportunities.

_Remediation:_ Implement input validation exactly as specified in this document (tables, syllable structure, hyphen grammar). When used in safety- or security-relevant workflows, consider checksuming or context-binding at the application layer (e.g., include context or MAC over the underlying binary) and use redundancy when verbally communicating critical values.

*Threat: Side-channel leakage via formatting.* The trailing-hyphen padding signal reveals the parity of the original octet length (odd/even).

_Remediation:_ Applications that consider length parity sensitive SHOULD avoid emitting the trailing-hyphen signal by ensuring even-length inputs or by wrapping Proquints inside a protected container. In most operational uses this leakage is not security-relevant.

*Threat: Injection into protocol or storage contexts.* Improper normalization or acceptance of characters outside the defined alphabet could enable injection or confusion in downstream systems.

_Remediation:_ Encoders SHOULD emit lowercase and MAY include interior hyphens for readability only. Decoders MUST accept case-insensitive input, MUST ignore interior hyphens, MUST enforce the defined tables and CVCVC pattern, and MUST reject leading hyphens, multiple trailing hyphens, and consecutive interior hyphens. Do not accept characters outside [a窶ｦ奴] and hyphen.

*Threat: Transport-specific risks.* Use of Proquints in URLs, filenames, DNS, or messaging systems can expose data if those channels are observable.

_Remediation:_ When Proquints convey sensitive data, use secure transport appropriate to the context, apply access control and retention policies to logs, and avoid transmitting sensitive values over unprotected voice or chat. Proquints are ASCII-only and hyphenated; they are generally safe for "LDH" contexts (letters窶ｦ電igits窶ｦ塗yphen), but applications MUST observe each system’s length and syntax limits.

## 9. IANA Considerations

This document has no IANA actions.

## 10. Process Note

The intent of this document is archival and implementer guidance via the Independent Stream. The author does not seek standardization at this time. If significant deployment or protocol integration interest emerges, a future effort MAY be dispatched within the IETF to consider Standards Track work.

## 11. Acknowledgments

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

The author also thanks Lucas Bremgartner for his detailed review and thoughtful suggestions. His insights substantially improved both the clarity and correctness of the specification. His independent implementation also provided a valuable cross-check of the design.

## 12. References

### 12.1. 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>.

### 12.2. 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