Network Working Group
Internet-Draft
Intended status: Informational
Expires: 6 May 2026

E. Davidson
FartLabs

2 November 2025

# A Deterministic Algorithm for Resolving Shortlinks with Internal Redirects
draft-go-protocol-00

## Abstract

This document specifies a deterministic algorithm for resolving shortlinks (compact string identifiers) into fully-qualified URLs. The algorithm supports both absolute URL destinations and origin-relative internal redirects with loop protection. It defines deterministic precedence rules for combining query parameters and fragments from multiple sources during resolution, enabling consistent behavior across clients, servers, and command-line tools.

## Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/EthanThatOneKid/go-protocol.

## 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 6 May 2026.

## Copyright Notice

Table of Contents

## 1. Introduction

Shortlinks provide a mechanism for mapping compact, memorable identifiers to longer URLs. However, without a standardized resolution algorithm, different implementations may resolve the same shortlink to different destinations or handle edge cases inconsistently, leading to interoperability issues.

This document specifies a deterministic algorithm for resolving shortlinks that:

* Supports longest-prefix key matching to enable hierarchical shortlink structures

* Handles both absolute URL destinations and origin-relative internal redirects

* Provides loop protection for internal redirect chains

* Defines clear precedence rules for combining query parameters and fragments from multiple sources

* Remains stateless and deterministic for consistent behavior across implementations

The algorithm is designed to work with any ruleset source (database, file, configuration, etc.) and requires no protocol-level changes or IANA registrations.

## 2. Conventions and Definitions

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.

The following terms are used throughout this document:

*Shortlink*: A compact string identifier that maps to a URL destination.

*Ruleset*: A collection of key-value pairs where keys are shortlink identifiers (non-empty strings without leading "/") and values are either absolute URLs (strings whose first four characters are "http") or origin-relative paths (strings whose first character is "/").

*Request URL*: The incoming URL to be resolved, which may contain an embedded shortlink in its pathname component.

*Destination URL*: The fully-qualified URL that results from resolving a shortlink.

*Internal Redirect*: A shortlink value that is an origin-relative path (whose first character is "/"), which requires further resolution within the same origin.

*Absolute Destination*: A shortlink value that is a fully-qualified URL (whose first four characters are "http", including both "http" and "https" schemes), which serves as the final destination.

*Longest-Prefix Match*: The process of finding the matching shortlink key by starting with the full pathname (minus leading "/") and progressively removing the trailing path segment (everything from the last "/" character onward, inclusive of the "/") until either a matching key is found in the Ruleset or no "/" character remains in the candidate key.

## 3. Algorithm Specification

### 3.1. Overview

The algorithm takes as input:

* A Request URL (a fully-qualified URL)

* A Ruleset (a collection of shortlink mappings)

The algorithm produces as output:

* A Destination URL (the resolved fully-qualified URL)

The algorithm operates in three main phases:

1. *Initialization*: Extract embedded query parameters and fragments from the Request URL

2. *Resolution*: Find the matching shortlink using longest-prefix matching, handling internal redirects as needed

3. *Construction*: Build the final Destination URL by combining components according to precedence rules

### 3.2. Initialization Phase

From the Request URL, extract and preserve the following components:

* *Origin*: The scheme, host, and port components of the Request URL

* *Request Query*: The search component of the Request URL (including the leading "?" if present, otherwise an empty string)

* *Request Fragment*: The hash component of the Request URL (including the leading "#" if present, otherwise an empty string)

* *Pathname*: The pathname component of the Request URL (the path portion, including the leading "/")

These preserved components will be used during final URL construction according to the precedence rules defined below. Initialize the following variables:

* relativePathname = "" (empty string)

* initialQuery = "" (empty string)

* initialHash = "" (empty string)

* redirectCount = 0

### 3.3. Resolution Phase

The resolution process begins with the pathname from the Request URL and searches for a matching shortlink key using longest-prefix matching.

#### 3.3.1. Longest-Prefix Matching

At the start of each resolution iteration, find a matching shortlink key using the following procedure:

1. Let candidate = pathname.slice(1) (the pathname with the leading "/" character removed). If pathname is exactly "/", then candidate = "" (empty string).

2. Check if candidate exists as a key in the Ruleset:

    * If yes, matchedKey = candidate, proceed to "Processing Matched Shortlinks"

    * If no, continue to step 3

3. While candidate.length > 0: a. Find the last occurrence of "/" in candidate using candidate.lastIndexOf("/") b. If no "/" is found (result is -1), exit the loop, no match found c. Set candidate = candidate.slice(0, slashIndex) (everything before the last "/", exclusive) d. Check if candidate exists as a key in the Ruleset:

   * If yes, matchedKey = candidate, proceed to "Processing Matched Shortlinks"

   * If no, repeat step 3

If no matching key is found after this process completes, the algorithm terminates and returns the Request URL with any accumulated relativePathname appended (see "No Match Found" below).

### 3.3.2. Handling Hash and Query in Pathname

At the start of each resolution iteration (before longest-prefix matching), if the pathname contains "#" or "?" characters, these MUST be extracted and removed from the pathname. The extraction MUST occur in the following order:

1. *Hash Extraction* (performed first):

   * Find the last occurrence of "#" in pathname using pathname.lastIndexOf("#")

   * If found (result >= 0):

     - If initialHash is empty (""), set initialHash = pathname.slice(hashIdx) (everything from "#" onward, inclusive)

     - Set pathname = pathname.slice(0, hashIdx) (everything before "#", exclusive)

2. *Query Extraction* (performed after hash extraction):

   * Find the last occurrence of "?" in the updated pathname using pathname.lastIndexOf("?")

   * If found (result >= 0):

     - If initialQuery is empty (""), set initialQuery = pathname.slice(queryIdx) (everything from "?" onward, inclusive)

- Set pathname = pathname.slice(0, queryIdx) (everything before "?", exclusive)

After extraction, the pathname contains only path segments and is used for longest-prefix matching. The extracted hash and query are preserved in initialHash and initialQuery (only the first extraction of each is preserved across iterations).

#### 3.3.3. Processing Matched Shortlinks

When a matching key matchedKey is found in the Ruleset:

1. Let value = the value associated with matchedKey in the Ruleset

2. Calculate the remaining pathname segment:

   * After removing the leading "/" from pathname and matching against matchedKey, the remaining segment is calculated as: remainingPath = pathname.slice(matchedKey.length + 1) where matchedKey.length + 1 accounts for the matched key and the "/" that follows it (or would follow it). This extracts everything from index matchedKey.length + 1 to the end of the pathname string.

   * If pathname.length <= matchedKey.length + 1, then remainingPath = "" (empty string), meaning the pathname ended exactly at or before the end of the matched key.

   * Prepend remainingPath to relativePathname: relativePathname = remainingPath + relativePathname (string concatenation, preserving any leading "/" in remainingPath)

3. Determine the value type:

   *Absolute Destination* (first four characters of value are "http"): - Parse value as a URL to obtain destination URL object - Resolution terminates immediately - Construct final URL (see "Construction Phase" below)

   *Internal Redirect* (first character of value is "/"): - Set pathname = value (the shortlink value becomes the new pathname) - Increment redirectCount by 1 - If redirectCount >= 256, terminate with error "too many internal redirects" - Continue to next resolution iteration (return to "Handling Hash and Query in Pathname" at the start of the iteration)

#### 3.3.4. No Match Found

If no matching key is found during longest-prefix matching:

* Construct a destination URL using new URL(pathname + relativePathname, origin) where origin is the Request URL origin
* This destination URL's pathname becomes the final destination pathname
* Proceed to "Construction Phase" with this destination URL

### 3.4. Construction Phase

After resolution completes (either by finding an absolute destination or no match), construct the final Destination URL using the following procedure:

#### 3.4.1. Base URL Construction

The base URL components are determined as follows:

* *Origin*:
  - If an absolute destination was found: use the origin from the parsed destination URL
  - Otherwise (no match found): use the Request URL origin
* *Pathname*:
  - If an absolute destination was found: start with the destination URL's pathname component (without leading/trailing modification)
  - Otherwise: use the pathname from the constructed URL in "No Match Found"
  - In both cases: append relativePathname directly (string concatenation) without any separator, as relativePathname already includes any necessary leading "/"

#### 3.4.2. Query Parameter Combination

Query parameters MUST be combined using the following procedure and precedence order (higher precedence overwrites lower precedence for matching parameter names):

1. *Base query*: Start with an empty URLSearchParams object

2. *Add destination query* (lowest precedence): If an absolute destination was found, parse destination.search (the destination URL's search component, including leading "?" if present) as URLSearchParams and add all parameters to the base

3. *Add pathname-embedded query* (medium precedence): If initialQuery is not empty, parse it (including leading "?" if present) as URLSearchParams and for each key-value pair, call URLSearchParams.set(key, value) on the base (this overwrites any existing parameters with the same key)

4. *Add Request URL query* (highest precedence): If requestQuery is not empty, parse it (including leading "?" if present) as URLSearchParams and for each key-value pair, call URLSearchParams.set(key, value) on the base (this overwrites any existing parameters with the same key)

Convert the final URLSearchParams to a string using URLSearchParams.toString(). If the result is non-empty, prefix it with "?" to form the final query string. Otherwise, the query string is "" (empty string).

#### 3.4.3. Fragment Selection

Fragment values MUST be selected using the following precedence (higher precedence replaces lower precedence):

1. *Request URL fragment* (highest precedence): If requestHash is not empty, use requestHash (which includes the leading "#")

2. *Pathname-embedded fragment* (medium precedence): If requestHash is empty and initialHash is not empty, use initialHash (which includes the leading "#")

3. *Destination URL fragment* (lowest precedence): If both requestHash and initialHash are empty and an absolute destination was found, use destination.hash (which includes the leading "#" if present, otherwise "")

4. *No fragment*: If none of the above apply, use "" (empty string)

Unlike query parameters, fragments are not merged; only one fragment value is used based on the precedence order above.

#### 3.4.4. Final URL Assembly

Assemble the final Destination URL by concatenating the following components in order:

{origin}{destination-pathname}{relativePathname}{query-string}{fragment-string}

Where:

* {origin} is the scheme, host, and port (e.g., "https://example.com")
* {destination-pathname} is the pathname component from the destination (e.g., "/path/to")
* {relativePathname} is the accumulated relative pathname (e.g., "/extra/segments")
* {query-string} is the combined query string from "Query Parameter Combination" (e.g., "?foo=bar" or "")
* {fragment-string} is the selected fragment from "Fragment Selection" (e.g., "#section" or "")

No additional separators are inserted between components beyond what is already present in the component values themselves.

### 3.5. Algorithm Pseudocode

The following pseudocode summarizes the algorithm:

‘‘‘ function resolve(RequestURL, Ruleset): // Initialize variables origin = RequestURL.origin requestQuery = RequestURL.search // Includes "?" if present, else "" requestHash = RequestURL.hash // Includes "#" if present, else "" pathname = RequestURL.pathname // Always starts with "/"

relativePathname = "" initialQuery = "" initialHash = "" redirectCount = 0 matchedKey = null // Initialize to null/undefined

while redirectCount < 256: matchedKey = null // Reset for each iteration

```
// Step 1: Extract embedded hash from pathname (first)
hashIdx = pathname.lastIndexOf("#")
if hashIdx >= 0:
  if initialHash == "":
    initialHash = pathname.slice(hashIdx)  // From "#" onward, inclusive
  pathname = pathname.slice(0, hashIdx)    // Before "#", exclusive

// Step 2: Extract embedded query from pathname (after hash)
queryIdx = pathname.lastIndexOf("?")
if queryIdx >= 0:
  if initialQuery == "":
    initialQuery = pathname.slice(queryIdx)  // From "?" onward, inclusive
  pathname = pathname.slice(0, queryIdx)     // Before "?", exclusive

// Step 3: Longest-prefix match
if pathname == "/":
  candidate = ""
else:
  candidate = pathname.slice(1)  // Remove leading "/"

// Try exact match first
if candidate != "" and Ruleset.hasOwnProperty(candidate):
  matchedKey = candidate
else:
  // Try progressively shorter prefixes
  while candidate.length > 0:
    slashIdx = candidate.lastIndexOf("/")
    if slashIdx == -1:
      break  // No more "/" found, no match
    candidate = candidate.slice(0, slashIdx)  // Remove last segment
    if Ruleset.hasOwnProperty(candidate):
      matchedKey = candidate
      break

// Step 4: Process match result
if matchedKey == null:
  // No match found
  destination = new URL(pathname + relativePathname, origin)
  query = combineQueries("", initialQuery, requestQuery)
  hash = selectFragment(requestHash, initialHash, "")
  return destination.origin + destination.pathname + query + hash

// Match found, process it
value = Ruleset[matchedKey]

// Calculate remaining path
if pathname.length > matchedKey.length + 1:
  remainingPath = pathname.slice(matchedKey.length + 1)
```

```
else:
  remainingPath = ""
relativePathname = remainingPath + relativePathname

// Check value type
if value.slice(0, 4) == "http":
  // Absolute destination
  destination = new URL(value)
  query = combineQueries(destination.search, initialQuery, requestQuery)
  hash = selectFragment(requestHash, initialHash, destination.hash)
  return destination.origin + destination.pathname +
         relativePathname + query + hash

if value.length > 0 and value[0] == "/":
  // Internal redirect
  pathname = value
  redirectCount++
  continue  // Next iteration

   // Loop limit exceeded throw Error("too many internal redirects")

   function combineQueries(baseQuery, pathnameQuery, requestQuery): //
   baseQuery is lowest precedence (destination query) // pathnameQuery
   is medium precedence // requestQuery is highest precedence

   params = new URLSearchParams(baseQuery)

   // Add pathname-embedded query (overwrites base) if pathnameQuery !=
   "": pathParams = new URLSearchParams(pathnameQuery) for each key-
   value pair (key, value) in pathParams: params.set(key, value)

   // Add request query (overwrites base and pathname) if requestQuery
   != "": requestParams = new URLSearchParams(requestQuery) for each
   key-value pair (key, value) in requestParams: params.set(key, value)

   result = params.toString() return result.length > 0 ? "?" + result :
   ""

   function selectFragment(requestHash, initialHash, destinationHash):
   if requestHash != "": return requestHash if initialHash != "": return
   initialHash return destinationHash // May be "" if destination has no
   fragment ‘’’
```

### 3.6. Example Resolutions

#### 3.6.1. Example 1: Absolute Destination with Path Appending

Ruleset:

json { "github": "https://github.com", "repo": "https://github.com/FartLabs/go" }

Request: https://example.com/github/ietf/guidelines

Resolution:

1. Pathname "/github/ietf/guidelines" - remove leading "/" to get candidate "github/ietf/guidelines"
2. Check Ruleset for "github/ietf/guidelines" - not found
3. Find last "/" at index 6, candidate becomes "github"
4. Check Ruleset for "github" - found: value "https://github.com"
5. Calculate remainingPath: pathname is "/github/ietf/guidelines", matchedKey is "github" (length 6), so pathname.slice(matchedKey.length + 1) = pathname.slice(7) = "/ietf/guidelines" (from index 7, which is the "/" after "github")
6. relativePathname = "/ietf/guidelines" + "" = "/ietf/guidelines"
7. Value starts with "http" → absolute destination
8. Parse "https://github.com" as URL: origin = "https://github.com", pathname = "/"
9. Query: "" (no queries), Fragment: "" (no fragments)
10. Result: https://github.com + / + /ietf/guidelines + + = https://github.com/ietf/guidelines

#### 3.6.2. Example 2: Internal Redirect Chain

Ruleset:

json { "docs": "/documentation", "docs/api": "/documentation/reference" }

Request: https://example.com/docs/api/v1/users

Resolution: *Iteration 1:*

1. Pathname "/docs/api/v1/users" - remove leading "/" to get candidate "docs/api/v1/users"

2. Check Ruleset for "docs/api/v1/users" - not found

3. Find last "/" at index 8, candidate becomes "docs/api"

4. Check Ruleset for "docs/api" - found: value "/documentation/reference"

5. Calculate remainingPath: pathname.slice(8 + 1) = pathname.slice(9) = "/v1/users"

6. relativePathname = "/v1/users" + "" = "/v1/users"

7. Value starts with "/" → internal redirect

8. pathname = "/documentation/reference", redirectCount = 1

*Iteration 2:*

1. Pathname "/documentation/reference" - remove leading "/" to get candidate "documentation/reference"

2. Check Ruleset for "documentation/reference" - not found

3. Find last "/" at index 13, candidate becomes "documentation"

4. Check Ruleset for "documentation" - not found

5. Find last "/" at index -1 (no "/" in "documentation") - no match found

6. Construct destination: new URL("/documentation/reference" + "/v1/users", origin)

7. Query: "" (no queries), Fragment: "" (no fragments)

8. Result: https://example.com/documentation/reference/v1/users

### 3.6.3. Example 3: Query Parameter Precedence

Ruleset:

json { "example": "https://example.com?baz=qux" }

Request: https://example.com/example?foo=bar

Resolution:

1. Pathname "/example" - remove leading "/" to get candidate "example"

2. Match found: key "example", value "https://example.com?baz=qux"

3. No remaining pathname (matchedKey.length + 1 = 8, pathname.length = 8, so remainingPath = "")

4. Value starts with "http" → absolute destination

5. Parse "https://example.com?baz=qux" as URL: destination.search = "?baz=qux"

6. Combine queries:

    * Base (destination): baz=qux

    * Pathname-embedded: (none, initialQuery = "")

    * Request: foo=bar (overwrites any existing foo, but baz remains)

    * Result: "?baz=qux&foo=bar"

7. Fragment: requestHash = "" (no fragment in request), initialHash = "", destinationHash = ""

8. Result: https://example.com/?baz=qux&foo=bar

#### 3.6.4. Example 4: Fragment Precedence

Ruleset:

json { "example": "https://example.com#yin" }

Request: https://example.com/example#yang

Resolution:

1. Pathname "/example" - remove leading "/" to get candidate "example"

2. Match found: key "example", value "https://example.com#yin"

3. No remaining pathname (matchedKey.length + 1 = 8, pathname.length = 8, so remainingPath = "")

4. Value starts with "http" → absolute destination

5. Parse "https://example.com#yin" as URL: destination.hash = "#yin"

6. Query: no query parameters present, result = ""

7. Fragment selection: requestHash = "#yang" (not empty) → use "#yang"

8. Result: https://example.com/#yang

#### 3.6.5. Example 5: No Match Returns Original

Ruleset:

json { "c": "https://example.com/calendar" }

Request: https://example.com/colors

Resolution:

1. Pathname "/colors" - remove leading "/" to get candidate "colors"

2. Check Ruleset for "colors" - not found

3. Find last "/" in "colors" - not found (lastIndexOf returns -1)

4. No matching key found

5. Construct destination: new URL("/colors" + "", origin) = "https://example.com/colors"

6. Query: "" (no queries), Fragment: "" (no fragments)

7. Result: https://example.com/colors (unchanged)

## 4. Security Considerations

### 4.1. Loop Protection

The algorithm includes protection against infinite redirect loops by limiting internal redirects to 256 iterations. Implementations MUST enforce this limit and MUST reject requests that exceed it. However, 256 may be insufficient for legitimate deeply-nested redirect structures; implementers should consider logging or monitoring redirect counts to identify potentially problematic rulesets.

### 4.2. Input Validation

Implementations SHOULD validate that:

* Request URLs are well-formed and use supported schemes
* Ruleset values are valid URLs or paths
* Pathname components do not contain illegal characters

Malformed input may cause unpredictable behavior or security vulnerabilities.

### 4.3. Ruleset Source Security

The security of the resolution algorithm depends on the trustworthiness of the ruleset source. Implementations SHOULD:

* Validate ruleset integrity (e.g., checksums, signatures)
* Restrict ruleset modifications to authorized entities
* Audit ruleset changes for suspicious patterns
* Implement rate limiting on resolution requests to prevent abuse

### 4.4. Information Disclosure

The resolution process may reveal information about the ruleset structure through timing differences or error messages. Implementations SHOULD consider:

* Using constant-time matching algorithms where timing attacks are a concern
* Providing uniform error responses that don’t reveal whether a shortlink key exists

### 4.5. Path Traversal

When appending relative pathname segments to destination URLs, implementations MUST ensure that the resulting pathname is safe and does not allow path traversal attacks (e.g., "../" sequences). The algorithm as specified does not explicitly sanitize these segments, so implementations should validate or sanitize the final constructed URL before use.

### 4.6. Query Parameter and Fragment Handling

The query parameter merging behavior means that user-supplied query parameters from the Request URL can overwrite destination URL parameters. This may be desired behavior, but implementers should be aware that it could be used to modify the intended destination behavior. Similarly, fragments from the Request URL take precedence, which may override intended anchor targets in the destination.

### 4.7. Redirect Chains

Even with loop protection, long redirect chains can be used for:

* Denial of service through excessive processing
* Obfuscation of final destination
* Tracking or analytics purposes

Implementations should consider logging redirect chains and potentially limiting chain length more aggressively than the 256-iteration limit.

## 5. IANA Considerations

This document has no IANA actions.

## 6. Normative References

[RFC2119] 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/rfc/rfc2119>.

[RFC8174] 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/rfc/rfc8174>.

## Appendix A. Acknowledgments

This document was inspired by the need for consistent shortlink resolution across different implementations and platforms.

## Author’s Address

Ethan Davidson
FartLabs
Email: ethan.r.davidson@gmail.com