Stream: Internet Engineering Task Force (IETF)
RFC: 9842
Category: Standards Track
Published: September 2025
ISSN: 2070-1721
Authors: P. Meenan, Ed. *Google LLC* Y. Weiss, Ed. *Shopify Inc.*

RFC 9842
Compression Dictionary Transport

Abstract

This document specifies a mechanism for dictionary-based compression in the Hypertext Transfer Protocol (HTTP). By utilizing this technique, clients and servers can reduce the size of transmitted data, leading to improved performance and reduced bandwidth consumption. This document extends existing HTTP compression methods and provides guidelines for the delivery and use of compression dictionaries within the HTTP protocol.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This specification defines a mechanism for using designated HTTP [HTTP] responses as an external dictionary for future HTTP responses for compression schemes that support using external dictionaries (e.g., Brotli [RFC7932] and Zstandard [ZSTD]).

This document describes the HTTP headers used for negotiating dictionary usage and registers content-encoding values for compressing with Brotli and Zstandard using a negotiated dictionary.

The negotiation of dictionary usage leverages HTTP's content negotiation (see Section 12 of [HTTP]) and is usable with all versions of HTTP.

1.1. Use Cases

Any HTTP response can be specified for use as a compression dictionary for future HTTP requests, which provides a lot of flexibility. Two common use cases that are seen frequently are described below.

1.1.1. Version Upgrade

Using a previous version of a resource as a dictionary for a newer version enables delivery of a delta-compressed version of the changes, usually resulting in significantly smaller responses than what can be achieved by compression alone.

For example:

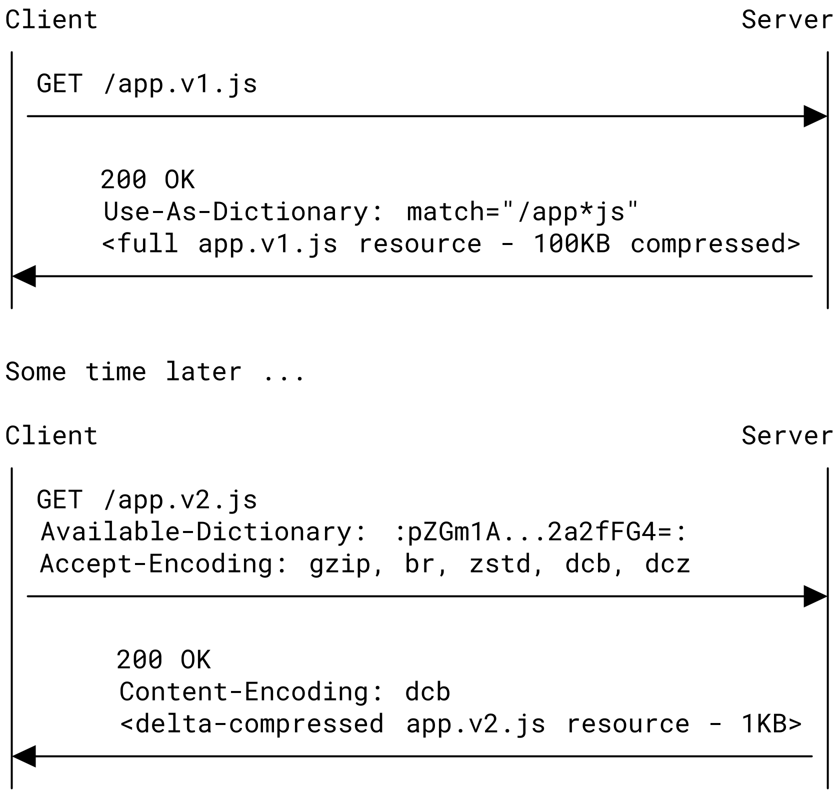

Figure 1: Version Upgrade Example

1.1.2. Common Content

If several resources share common patterns in their responses, then it can be useful to reference an external dictionary that contains those common patterns, effectively compressing them out of the responses. Some examples of this are common template HTML for similar pages across a site and common keys and values in API calls.

For example:

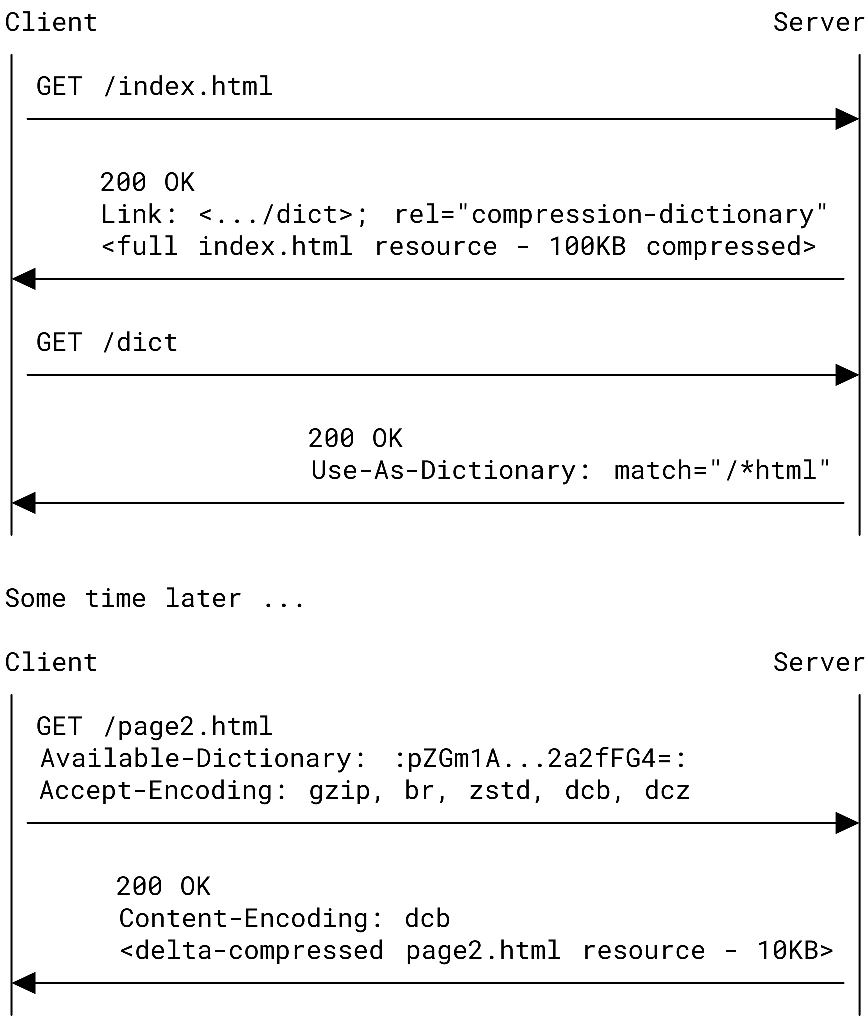

Figure 2: Common Content Example

1.2. Notational Conventions

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 uses the following terminology from Section 3 of [STRUCTURED-FIELDS] to specify syntax and parsing: Dictionary, String, Inner List, Token, and Byte Sequence.

This document uses the line folding strategies described in [FOLDING].

This document also uses terminology from [HTTP] and [HTTP-CACHING].

2. Dictionary Negotiation

2.1. Use-As-Dictionary

When responding to an HTTP Request, a server can advertise that the response can be used as a dictionary for future requests for URLs that match the rules specified in the "Use-As-Dictionary" response header.

The "Use-As-Dictionary" response header is a Structured Field [STRUCTURED-FIELDS] Dictionary with values for "match", "match-dest", "id", and "type".

2.1.1. "match"

The "match" value of the "Use-As-Dictionary" response header is a String value that provides the URL Pattern to use for request matching (see [URLPATTERN]).

The URL Pattern used for matching does not support using regular expressions.

The following algorithm is used to validate that a given String value is a valid URL Pattern that does not use regular expressions and is for the same Origin (Section 4.3.1 of [HTTP]) as the dictionary. It will return TRUE for a valid match pattern and FALSE for an invalid pattern that **MUST NOT** be used.

1. Let MATCH be the value of "match" for the given dictionary.
2. Let URL be the URL of the dictionary request.
3. Let PATTERN be a "URL pattern struct" created by running the steps to "create a URL pattern" by setting input=MATCH and baseURL=URL (see Section 1.4 of [URLPATTERN]).
4. If the result of running the "has regexp groups" steps for PATTERN returns TRUE, then return FALSE (see the last list in Section 1.4 of [URLPATTERN]).
5. Return TRUE.

The "match" value is required and **MUST** be included in the "Use-As-Dictionary" response header for the dictionary to be considered valid.

Operating at the HTTP level, the specified match patterns will operate on the percent-encoded version of the URL path (see Section 2 of [URL]).

For example, the URL "http://www.example.com/düsseldorf" would be encoded as "http://www.example.com/d%C3%BCsseldorf" and a relevant match pattern would be:

```
Use-As-Dictionary: match="/d%C3%BCsseldorf"
```

2.1.2. "match-dest"

The "match-dest" value of the "Use-As-Dictionary" response header is an Inner List of String values that provides a list of Fetch request destinations for the dictionary to match (see "RequestDestination" in Section 5.4 of [FETCH]).

An empty list for "match-dest" **MUST** match all destinations.

For clients that do not support request destinations, the client **MUST** treat it as an empty list and match all destinations.

The "match-dest" value is optional and defaults to an empty list.

2.1.3. "id"

The "id" value of the "Use-As-Dictionary" response header is a String value that specifies a server identifier for the dictionary. If an "id" value is present and has a string length longer than zero, then it **MUST** be sent to the server in a "Dictionary-ID" request header when the client sends an "Available-Dictionary" request header for the same dictionary (see Section 2.2).

The server identifier **MUST** be treated as an opaque string by the client.

The server identifier **MUST NOT** be relied upon by the server to guarantee the contents of the dictionary. The dictionary hash **MUST** be validated before use.

The "id" value string length (after any decoding) supports up to 1024 characters.

The "id" value is optional and defaults to the empty string.

2.1.4. "type"

The "type" value of the "Use-As-Dictionary" response header is a Token value that describes the file format of the supplied dictionary.

"raw" is defined as a dictionary format that represents an unformatted blob of bytes suitable for any compression scheme to use.

If a client receives a dictionary with a type that it does not understand, it **MUST NOT** use the dictionary.

The "type" value is optional and defaults to "raw".

2.1.5. Examples

2.1.5.1. Path Prefix

A response that contained a response header (as shown below) would specify matching any document request for a URL with a path prefix of /product/ on the same Origin (Section 4.3.1 of [HTTP]) as the original request:

```
NOTE: '\' line wrapping per RFC 8792

Use-As-Dictionary: \
  match="/product/*", match-dest=("document")
```

2.1.5.2. Versioned Directories

A response that contained a response header (as shown below) would match any path that starts with "/app/" and ends with "/main.js":

```
Use-As-Dictionary: match="/app/*/main.js"
```

2.2. Available-Dictionary

When an HTTP client makes a request for a resource for which it has an appropriate dictionary, it can add an "Available-Dictionary" request header to the request to indicate to the server that it has a dictionary available to use for compression.

The "Available-Dictionary" request header is a Structured Field [STRUCTURED-FIELDS] Byte Sequence containing the SHA-256 [SHA-256] hash of the contents of a single available dictionary.

The client **MUST** only send a single "Available-Dictionary" request header with a single hash value for the best available match that it has available.

For example:

```
Available-Dictionary: :pZGm1Av0IEBKARczz7exkNYsZb8LzaMrV7J32a2fFG4=:
```

2.2.1. Dictionary Freshness Requirement

To be considered as a match, the dictionary resource **MUST** be either fresh [HTTP-CACHING] or allowed to be served stale (see [RFC5861]).

2.2.2. Dictionary URL Matching

When a dictionary is stored as a result of a "Use-As-Dictionary" directive, it includes a "match" string and an optional "match-dest" string that are used to match an outgoing request from a client to the available dictionaries.

To see if an outbound request matches a given dictionary, the following algorithm will return TRUE for a successful match and FALSE for no-match:

1. If the current client supports request destinations and a "match-dest" string was provided with the dictionary:
   - Let DEST be the value of "match-dest" for the given dictionary.
   - Let REQUEST_DEST be the value of the destination for the current request.

   - If DEST is not an empty list and if REQUEST_DEST is not in the DEST list of destinations, return FALSE.

2. Let BASEURL be the URL of the dictionary request.
3. Let URL represent the URL of the outbound request being checked.
4. If the Origin of BASEURL and the Origin of URL are not the same, return FALSE (see Section 4.3.1 of [HTTP]).
5. Let MATCH be the value of "match" for the given dictionary.
6. Let PATTERN be a "URL pattern struct" created by running the steps to "create a URL pattern" by setting input=MATCH and baseURL=URL (see Section 1.4 of [URLPATTERN]).
7. Return the result of running the "match" steps on PATTERN with input=URL, which will check for a match between the request URL and the supplied "match" string (see "Match" in Section 1.4 of [URLPATTERN]).

2.2.3. Multiple Matching Dictionaries

When there are multiple dictionaries that match a given request URL, the client **MUST** pick a single dictionary using the following rules:

1. For clients that support request destinations, a dictionary that specifies and matches a "match-dest" takes precedence over a match that does not use a destination.
2. Given equivalent destination precedence, the dictionary with the longest "match" takes precedence.
3. Given equivalent destination and match length precedence, the most recently fetched dictionary takes precedence.

2.3. Dictionary-ID

When an HTTP client makes a request for a resource for which it has an appropriate dictionary and the dictionary was stored with a server-provided "id" in the "Use-As-Dictionary" response header, the client **MUST** echo the stored "id" in a "Dictionary-ID" request header.

The "Dictionary-ID" request header is a Structured Field [STRUCTURED-FIELDS] String of up to 1024 characters (after any decoding) and **MUST** be identical to the server-provided "id".

For example, given an HTTP response that set a dictionary ID:

```
Use-As-Dictionary: match="/app/*/main.js", id="dictionary-12345"
```

A future request that matches the given dictionary will include both the hash and the ID:

```
Available-Dictionary: :pZGm1Av0IEBKARczz7exkNYsZb8LzaMrV7J32a2fFG4=:
Dictionary-ID: "dictionary-12345"
```

3. The "compression-dictionary" Link Relation Type

This specification defines the "compression-dictionary" link relation type [WEB-LINKING] that provides a mechanism for an HTTP response to provide a URL for a compression dictionary that is related to but not directly used by the current HTTP response.

The "compression-dictionary" link relation type indicates that fetching and caching the specified resource is likely to be beneficial for future requests. The response to some of those future requests likely have the ability to use the indicated resource as a compression dictionary.

Clients can fetch the provided resource at a time that they determine would be appropriate.

The response to the fetch for the compression dictionary needs to include a "Use-As-Dictionary" response header and a caching response header for it to be usable as a compression dictionary. The link relation only provides the mechanism for triggering the fetch of the dictionary.

The following example shows a link to a resource at https://example.org/dict.dat that is expected to produce a compression dictionary:

```
Link: <https://example.org/dict.dat>; rel="compression-dictionary"
```

4. Dictionary-Compressed Brotli

The "dcb" content encoding identifies a resource that is a "Dictionary-Compressed Brotli" stream.

A "Dictionary-Compressed Brotli" stream has a fixed header that is followed by a Shared Brotli [SHARED-BROTLI] stream. The header consists of a fixed 4-byte sequence and a 32-byte hash of the external dictionary that was used. The Shared Brotli stream is created using the referenced external dictionary and a compression window that is at most 16 MB in size.

The dictionary used for the "dcb" content encoding is a "raw" dictionary type as defined in Section 2.1.4 and is treated as a prefix dictionary as defined in Section 8.2 of [SHARED-BROTLI].

The 36-byte fixed header is as follows:

Magic_Number: 4 fixed bytes -- 0xff, 0x44, 0x43, 0x42.

SHA_256_Hash: 32 bytes. SHA-256 hash digest of the dictionary [SHA-256].

Clients that announce support for dcb content encoding **MUST** be able to decompress resources that were compressed with a window size of up to 16 MB.

With Brotli compression, the full dictionary is available during compression and decompression independent of the compression window, allowing for delta-compression of resources larger than the compression window.

5. Dictionary-Compressed Zstandard

The "dcz" content encoding identifies a resource that is a "Dictionary-Compressed Zstandard" stream.

A "Dictionary-Compressed Zstandard" stream is a binary stream that starts with a 40-byte fixed header and is followed by a Zstandard [ZSTD] stream of the response that has been compressed with an external dictionary.

The dictionary used for the "dcz" content encoding is a "raw" dictionary type as defined in Section 2.1.4 and is treated as a raw dictionary as per Section 5 of [ZSTD].

The 40-byte header consists of a fixed 8-byte sequence followed by the 32-byte SHA-256 hash of the external dictionary that was used to compress the resource:

Magic_Number: 8 fixed bytes -- 0x5e, 0x2a, 0x4d, 0x18, 0x20, 0x00, 0x00, 0x00.

SHA_256_Hash: 32 bytes. SHA-256 hash digest of the dictionary [SHA-256].

The 40-byte header is a Zstandard skippable frame (little-endian 0x184D2A5E) with a 32-byte length (little-endian 0x00000020) that is compatible with existing Zstandard decoders.

Clients that announce support for dcz content encoding **MUST** be able to decompress resources that were compressed with a window size of at least 8 MB or 1.25 times the size of the dictionary, whichever is greater, up to a maximum of 128 MB.

The window size used will be encoded in the content (currently, this can be expressed in powers of two only) and it **MUST** be lower than this limit. An implementation **MAY** treat a window size that exceeds the limit as a decoding error.

With Zstandard compression, the full dictionary is available during compression and decompression until the size of the input exceeds the compression window. Beyond that point, the dictionary becomes unavailable. Using a compression window that is 1.25 times the size of the dictionary allows for full delta compression of resources that have grown by 25% between releases while still giving the client control over the memory it will need to allocate for a given response.

6. Negotiating the Content Encoding

When a compression dictionary is available to compress the response to a given request, the encoding to be used is negotiated through the regular mechanism for negotiating content encoding in HTTP through the "Accept-Encoding" request header and "Content-Encoding" response header.

The dictionary to use is negotiated separately and advertised in the "Available-Dictionary" request header.

6.1. Accept-Encoding

When a dictionary is available for use on a given request and the client chooses to make dictionary-based content encoding available, the client adds the dictionary-aware content encodings that it supports to the "Accept-Encoding" request header. For example:

```
Accept-Encoding: gzip, deflate, br, zstd, dcb, dcz
```

When a client does not have a stored dictionary that matches the request or chooses not to use one for the request, the client **MUST NOT** send its dictionary-aware content encodings in the "Accept-Encoding" request header.

6.2. Content-Encoding

If a server supports one of the dictionary encodings advertised by the client and chooses to compress the content of the response using the dictionary that the client has advertised, then it sets the "Content-Encoding" response header to the appropriate value for the algorithm selected. For example:

```
Content-Encoding: dcb
```

If the response is cacheable, it **MUST** include a "Vary" header to prevent caches from serving dictionary-compressed resources to clients that don't support them or serving the response compressed with the wrong dictionary. For example:

```
Vary: accept-encoding, available-dictionary
```

7. IANA Considerations

7.1. Content Encoding Registration

IANA has added the following entries to the "HTTP Content Coding Registry" maintained at <https://www.iana.org/assignments/http-parameters/>:

Name: dcb
Description: "Dictionary-Compressed Brotli" data format.
Reference: RFC 9842, Section 4

Name: dcz
Description: "Dictionary-Compressed Zstandard" data format.
Reference: RFC 9842, Section 5

7.2. Header Field Registration

IANA has added the following entries to the "Hypertext Transfer Protocol (HTTP) Field Name Registry" maintained at <https://www.iana.org/assignments/http-fields/>:

Field Name	Status	Reference
Use-As-Dictionary	permanent	RFC 9842, Section 2.1
Available-Dictionary	permanent	RFC 9842, Section 2.2
Dictionary-ID	permanent	RFC 9842, Section 2.3

Table 1

7.3. Link Relation Registration

IANA has added the following entry to the "Link Relation Types" registry maintained at <https://www.iana.org/assignments/link-relations/>:

Relation Name: compression-dictionary
Description: Refers to a compression dictionary used for content encoding.
Reference: RFC 9842, Section 3

8. Compatibility Considerations

It is not unusual for devices to be on the network path that intercept, inspect, and process HTTP requests (web proxies, firewalls, intrusion detection systems, etc.). To minimize the risk of these devices incorrectly processing dictionary-compressed responses, compression dictionary transport **MUST** only be used in secure contexts (HTTPS).

9. Security Considerations

The security considerations for Brotli [RFC7932], Shared Brotli [SHARED-BROTLI], and Zstandard [ZSTD] apply to the dictionary-based versions of the respective algorithms.

9.1. Changing Content

The dictionary must be treated with the same security precautions as the content because a change to the dictionary can result in a change to the decompressed content.

The dictionary is validated using an SHA-256 hash of the content to make sure that the client and server are both using the same dictionary. The strength of the SHA-256 hash algorithm isn't explicitly needed to counter attacks since the dictionary is being served from the same origin as the content. That said, if a weakness is discovered in SHA-256 and it is determined that the dictionary negotiation should use a different hash algorithm, the "Use-As-Dictionary" response header can be extended to specify a different algorithm and the server would just ignore any "Available-Dictionary" requests that do not use the updated hash.

9.2. Reading Content

The compression attacks in Section 2.6 of [RFC7457] show that it's a bad idea to compress data from mixed (e.g., public and private) sources. The data sources include not only the compressed data but also the dictionaries. For example, if secret cookies are compressed using a public-data-only dictionary, information about the cookies is still leaked.

The dictionary can reveal information about the compressed data and vice versa. That is, data compressed with the dictionary can reveal contents of the dictionary when an adversary can control parts of the data to compress and see the compressed size. On the other hand, if the adversary can control the dictionary, the adversary can learn information about the compressed data.

9.3. Security Mitigations

If any of the mitigations do not pass, the client **MUST** drop the response and return an error.

9.3.1. Cross-Origin Protection

To make sure that a dictionary can only impact content from the same origin where the dictionary was served, the URL Pattern used for matching a dictionary to requests (Section 2.1.1) is guaranteed to be for the same origin that the dictionary is served from.

9.3.2. Response Readability

For clients, like web browsers, that provide additional protection against the readability of the payload of a response and against user tracking, additional protections **MUST** be taken to make sure that the use of dictionary-based compression does not reveal information that would not otherwise be available.

In these cases, dictionary compression **MUST** only be used when both the dictionary and the compressed response are fully readable by the client.

In browser terms, that means either the dictionary and compressed response are same-origin to the context they are being fetched from or the response is cross-origin and passes the Cross-Origin Resource Sharing (CORS) check (see Section 4.9 of [FETCH]).

9.3.3. Server Responsibility

As with any usage of compressed content in a secure context, a potential timing attack exists if the attacker can control any part of the dictionary or if it can read the dictionary and control any part of the content being compressed while performing multiple requests that vary the dictionary or injected content. Under such an attack, the changing size or processing time of the response reveals information about the content, which might be sufficient to read the supposedly secure response.

In general, a server can mitigate such attacks by preventing variations per request, as in preventing active use of multiple dictionaries for the same content, disabling compression when any portion of the content comes from uncontrolled sources, and securing access and control over the dictionary content in the same way as the response content. In addition, the following requirements on a server are intended to disable dictionary-aware compression when the client provides CORS request header fields that indicate a cross-origin request context.

The following algorithm will return FALSE for cross-origin requests where precautions such as not using dictionary-based compression should be considered:

1. If there is no "Sec-Fetch-Site" request header, return TRUE.
2. If the value of the "Sec-Fetch-Site" request header is "same-origin", return TRUE.
3. If there is no "Sec-Fetch-Mode" request header, return TRUE.
4. If the value of the "Sec-Fetch-Mode" request header is "navigate" or "same-origin", return TRUE.
5. If the value of the "Sec-Fetch-Mode" request header is "cors":
   - If the response does not include an "Access-Control-Allow-Origin" response header, return FALSE.
   - If the request does not include an "Origin" request header, return FALSE.
   - If the value of the "Access-Control-Allow-Origin" response header is "*", return TRUE.
   - If the value of the "Access-Control-Allow-Origin" response header matches the value of the "Origin" request header, return TRUE.
6. Return FALSE.

10. Privacy Considerations

Since dictionaries are advertised in future requests using the hash of the content of the dictionary, it is possible to abuse the dictionary to turn it into a tracking cookie.

To mitigate any additional tracking concerns, clients **MUST** treat dictionaries in the same way that they treat cookies [RFC6265]. This includes partitioning the storage using partitioning similar to or stricter than the partitioning used for cookies, as well as clearing the dictionaries whenever cookies are cleared.

11. References

11.1. Normative References

[FETCH] WHATWG, "Fetch Standard", WHATWG Living Standard, <https://fetch.spec.whatwg.org/>. Commit snapshot: <https://fetch.spec.whatwg.org/commit-snapshots/5a9680638ebfc2b3b7f4efb2bef0b579a2663951/>

[FOLDING] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[HTTP-CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[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/info/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/info/rfc8174>.

[SHA-256] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[SHARED-BROTLI] Alakuijala, J., Duong, T., Kliuchnikov, E., Szabadka, Z., and L. Vandevenne, "Shared Brotli Compressed Data Format", RFC 9841, DOI 10.17487/RFC9841, September 2025, <https://www.rfc-editor.org/info/rfc9841>.

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

[URL] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[URLPATTERN] WHATWG, "URL Pattern Standard", WHATWG Living Standard, <https://urlpattern.spec.whatwg.org/>. Commit snapshot: <https://urlpattern.spec.whatwg.org/commit-snapshots/696b4029d52e5854044bac6b72cdb198cb962ed0/>

[WEB-LINKING] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[ZSTD] Collet, Y. and M. Kucherawy, Ed., "Zstandard Compression and the 'application/zstd' Media Type", RFC 8878, DOI 10.17487/RFC8878, February 2021, <https://www.rfc-editor.org/info/rfc8878>.

11.2. Informative References

[RFC5861] Nottingham, M., "HTTP Cache-Control Extensions for Stale Content", RFC 5861, DOI 10.17487/RFC5861, May 2010, <https://www.rfc-editor.org/info/rfc5861>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC7457] Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)", RFC 7457, DOI 10.17487/RFC7457, February 2015, <https://www.rfc-editor.org/info/rfc7457>.

[RFC7932] Alakuijala, J. and Z. Szabadka, "Brotli Compressed Data Format", RFC 7932, DOI 10.17487/RFC7932, July 2016, <https://www.rfc-editor.org/info/rfc7932>.

Authors' Addresses

Patrick Meenan (EDITOR)
Google LLC
Email: pmeenan@google.com

Yoav Weiss (EDITOR)
Shopify Inc.
Email: yoav.weiss@shopify.com