

Content Delivery Networks Interconnection
Internet-Draft
Intended status: Standards Track
Expires: 2 January 2026

W. Power
G. Goldstein
Lumen Technologies
1 July 2025

CDNI Cache Control Metadata
draft-ietf-cdni-cache-control-metadata-04

Abstract

This specification adds new Cache Control objects that complement the basic Cache Control Metadata object defined in RFC8006, providing content providers and upstream Content Delivery Networks (uCDNs) more fine-grained control over downstream CDN (dCDN) caching. Use cases include overriding or adjusting cache control headers from the Content Service Provider (CSP) source or origin, bypassing caching altogether, or altering cache keys with dynamically generated values.

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 2 January 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. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
2. Requirements	3
3. Metadata Model Objects	3
3.1. MI.CachePolicy	3
3.2. MI.NegativeCachePolicy	6
3.3. MI.StaleContentCachePolicy	7
3.4. MI.CacheBypassPolicy	9
3.5. MI.ComputedCacheKey	10
4. Informative Examples	11
5. Security Considerations	13
6. IANA Considerations	13
6.1. CDNI Payload Types	13
7. Acknowledgements	14
8. Normative References	14
9. Informative References	15
Authors' Addresses	15

1. Introduction

In addition to the cache control parameters currently specified by the Cache object in Section 4.2.6 of [RFC8006], content providers and uCDNs often need more fine-grained control over dCDN caching, including scenarios where it is desirable to override or adjust cache control headers from the content provider's upstream source (aka origin). Commercial CDNs and Open Caching systems, which serve the role as dCDNs, require cache control instructions from the content provider or uCDN to cache the content according to the content provider's original intent. Within the context of CDNi, Open Caching systems are essentially dCDNs that use CDNi standards for configuration and traffic delegation.

The following capabilities are required for content providers and uCDNs to express cache control details to dCDNs:

- * Positive Cache Control - Allows the uCDN to specify both internal and external caching policies for the dCDN, where internal policies dictate how the dCDN manages its internal caching rules and the external policies dictate the cache controls headers that the dCDNs transmits to the User Agent (client).
- * Negative Cache Control - Allows the uCDN to specify caching policies based on error response codes received from the upstream source. Typical use case is for fine-grained control of the downstream and User Agent caching of error responses. For example, it may be desirable to cache error responses at the dCDN for a short period of time to prevent an overwhelmed origin source service or uCDN from being flooded with requests.
- * Cache Bypass Control - Allows content providers to bypass CDN caching when needed (typically for testing or performance benchmarking purposes).
- * Stale Content Policies - Allows control over how the dCDN should process requests for stale content. For example, this policy allows the content provider to specify that stale content be served from cache for a specified time period while refreshes from the origin source occur asynchronously.
- * Dynamically Constructed Cache Keys - It is typical in advanced CDN configurations to generate cache keys that are dynamically constructed via lightweight processing of various properties of the Hypertext Transfer Protocol (HTTP) request and/or response. As an example, an upstream source may specify a cache key as a value returned in a specific HTTP response header. The Metadata Expression Language (MEL) [I-D.power-metadata-expression-language] is used to allow for such advanced cache key construction.

2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Metadata Model Objects

3.1. MI.CachePolicy

MI.CachePolicy is a new GenericMetadata object that allows the uCDN to specify internal caching policies for the dCDN, as well as external caching policies expressed to clients of the dCDN (overriding any cache control policy set in the response from the uCDN).

The enumerated options for internal and external cache policies are defined as follows:

- * **as-is:** The freshness value of the object is computed at the acquisition/validation moment within the dCDN based solely on the uCDN upstream source response. This definition does not mandate any specific implementation for the dCDN for propagating content expiration and freshness values to the client (User Agent). Different dCDNs MAY use different sets of response headers according to their type and implementation, as long as these implementations result in equivalent caching and expiration behaviors.
- * **no-cache:** Indicates that a stored response SHOULD NOT be used to satisfy the request without successful validation on the upstream source server. See [RFC9111]
- * **no-store:** Indicates that a cache MUST NOT store any part of either this request or any response to it. See [RFC9111]

Property: internal

- * **Description:** Specifies the internal cache control policy that MUST be used by the dCDN.
- * **Type:** Either an integer in seconds (e.g., 5 for a five-second cache) or one of these enumerated options: "as-is", or "no-cache" or "no-store" as described above.
- * **Mandatory-to-Specify:** No. The default is "as-is", regardless of whether force-internal is unspecified, "True", or "False".

Property: external

- * **Description:** Specifies the external cache control policy that MUST be expressed to clients of the dCDN.
- * **Type:** String. Either an integer in seconds (e.g., 5 for a five-second cache) or one of these enumerated options: "as-is", or "no-cache" or "no-store" as described above.
- * **Mandatory-to-Specify:** No. The default is "as-is", regardless of whether force-external is unspecified, "True", or, "False".

Property: force-internal

- * Description: If set to "True", the metadata interface cache policy defined in the MI.CachePolicy internal property value will override any cache control policy set in the response from the uCDN. If set to "False", the MI.CachePolicy internal property value is only used if there is no cache control policy provided in the response from the uCDN.
- * Type: Boolean
- * Mandatory-to-Specify: No. The default is "False", which will apply the MI.CachePolicy internal property value only if no policy is provided in the response from the uCDN.

Property: force-external

- * Description: If set to "True", the metadata interface cache policy defined in the MI.CachePolicy external property value will override any cache control policy set in the response from the uCDN. If set to "False", the MI.CachePolicy external property value is only used if there is no cache control policy provided in the response from the uCDN.
- * Type: Boolean
- * Mandatory-to-Specify: No. The default is "False", which will apply the MI.CachePolicy external property value only if no policy is provided in the response from the uCDN.

When MI.CachePolicy is not specified, the dCDN will follow the uCDN expiration and freshness response directives. In case the uCDN did not send any such directives, the dCDN will follow its own policies.

In the following example, an MI.CachePolicy sets the internal cache control policy to five seconds. The external cache policy is set to 'no-cache' and both policies are forced:

```
{
  "generic-metadata-type": "MI.CachePolicy",
  "generic-metadata-value": {
    "internal": 5,
    "external": "no-cache",
    "force-internal": true,
    "force-external": true
  }
}
```

Figure 1

In the following example,, an MI.CachePolicy sets the internal cache control policy to "as-is" (keep the policy set in the response from the uCDN). The external cache policy is set to 'no-cache' and forced to override any cache control policy set in the response from the upstream source or uCDN:

```
{
  "generic-metadata-type": "MI.CachePolicy",
  "generic-metadata-value": {
    "internal": "as-is",
    "external": "no-cache",
    "force-external": true
  }
}
```

Figure 2

3.2. MI.NegativeCachePolicy

MI.NegativeCachePolicy is a new GenericMetadata object that allows the specification of caching policies based on HTTP response codes received from the upstream source.

Property: error-codes

- * Description: An array of HTTP response status codes (see Sections 15.5 and 15.6 of [RFC9110]), that, if returned from the uCDN, will be cached using the cache policy defined by the cache-policy property.
- * Type: Array of HTTP response status codes encoded as strings. Any HTTP status code from 100 to 599, or any of the special values, "2xx", "3xx", "4xx" or "5xx", where "xx" implies everything from 00 to 99. The use of 4xx, for example, would specify that 416 responses are cached. While repeated and redundant values in the array are allowed, they SHOULD be avoided for efficiency (no reason to specify both 5xx and 503, for example).
- * Mandatory-to-Specify: No. The default is to revert to [RFC8006] behavior. An empty or unspecified list MAY be used as a means to revoke a list inherited from an upper-level configuration.

Property: cache-policy

- * Description: The MI.CachePolicy to apply to the HTTP response status codes returned by the uCDN.
- * Mandatory-to-Specify: Yes

In the following example, a `MI.NegativeCachePolicy` object applies a no-cache policy whenever a 403 error code or any 5xx error code are seen from the upstream source.

```
{
  "generic-metadata-type": "MI.NegativeCachePolicy",
  "generic-metadata-value": {
    "error-codes": [ "403", "5xx" ],
    "cache-policy": {
      "internal": 5,
      "external": "no-cache",
      "force-internal": true,
      "force-external": true
    }
  }
}
```

Figure 3

3.3. `MI.StaleContentCachePolicy`

`MI.StaleContentCachePolicy` is a new `GenericMetadata` object that allows the uCDN to specify the policy the dCDN MUST use when responding with stale content. For example, this policy allows the content provider to specify that stale content MUST only be served from cache for a specified time period while the content is revalidated asynchronously.

Property: `stale-while-revalidating`

- * Description: Instructs the dCDN to serve a stale version of a resource while revalidating the resource with the uCDN. When set to "True", the dCDN will return a previously cached version of a resource while the resource is revalidated with the uCDN in the background.
- * Type: Boolean
- * Mandatory-to-Specify: No. The default is "False", which waits for the uCDN to revalidate a resource before responding to the client.

Property: `stale-if-error`

- * Description: Instructs the dCDN to serve a stale version of a resource if any one of a specified set of HTTP status codes was received when trying to revalidate the resource with the uCDN. In this case, the dCDN will return a previously cached version of a resource instead of caching the error response. While this

capability is typically used for well-understood HTTP error status codes, a list of any HTTP codes can be provided for maximum flexibility.

- * Type: Array of HTTP response status codes encoded as strings.. Any HTTP status code from 100 to 599, or any of the special values "2xx", "3xx", "4xx" or "5xx", where "xx" implies everything from 00 to 99. While repeated and redundant values in the array are allowed, they SHOULD be avoided for efficiency (no reason to specify both 5xx and 503, for example).
- * Mandatory-to-Specify: No. The default is to not serve stale content. An empty or unspecified list MAY be used as a means to revoke a list inherited from an upper-level configuration.

Property: failed-revalidation-delta-seconds

- * Description: Drives the dCDN behavior when revalidation attempts to the uCDN fail, specifying the number of seconds to wait before making another attempt to revalidate the resource with the uCDN. Use of failed-revalidation-delta-seconds allows load to be reduced on the uCDN during times of system stress. Stale content is served during this wait period.
- * Type: Integer (seconds)
- * Mandatory-to-Specify: No. The default is zero, which means an attempt will be made to revalidate the resource with the uCDN immediately.

In the following example , an MI.StaleContentCachePolicy where stale-while-revalidating is true instructs the dCDN to respond with a stale cached version of the resource while it revalidates the resource with the uCDN in the background:

```
{
  "generic-metadata-type": "MI.StaleContentCachePolicy",
  "generic-metadata-value": {
    "stale-while-revalidating": true
  }
}
```

Figure 4

In the following example,, an MI.StaleContentCachePolicy where stale-if-error instructs the dCDN to use the stale cached resource if it receives an error of type 503 or 504 when trying to revalidate the resource with the uCDN.


```
{
  "generic-metadata-type": "MI.StaleContentCachePolicy",
  "generic-metadata-value": {
    "stale-if-error": [ "503", "504" ]
  }
}
```

Figure 5

In the following example, an `MI.StaleContentCachePolicy` where `stale-while-revalidating` is true instructs the dCDN to respond with a stale cached version of the resource while it revalidates the resource with the uCDN in the background.

- * `stale-if-error` instructs the dCDN to use the stale cached resource if it receives an error of type 404 or any 5xx status when trying to revalidate the resource with the uCDN.
- * `failed-revalidation-delta-seconds` instructs the dCDN to use an additional five-second cache time-to-live (TTL) before making another attempt to revalidate the resource from the uCDN. That is, the dCDN will serve (the now stale) resource for another five seconds before making another attempt to revalidate the resource with the uCDN.

```
{
  "generic-metadata-type": "MI.StaleContentCachePolicy",
  "generic-metadata-value": {
    "stale-while-revalidating": true,
    "stale-if-error": [ "404", "5xx" ],
    "failed-revalidation-delta-seconds": 5
  }
}
```

Figure 6

3.4. `MI.CacheBypassPolicy`

`MI.CacheBypassPolicy` is a new `GenericMetadata` object that allows a client request to be set as non-cacheable. It is expected that this feature will be used to allow clients to bypass caching when testing the uCDN fill path. Note: `MI.CacheBypassPolicy` is typically used in conjunction with a path match or match expression on a header value or query parameter. Any content previously cached (by client requests that do not set `MI.CacheBypassPolicy`) MUST NOT be evicted.

Property: `bypass-cache`

- * Description: A Boolean value that can activate the feature for a given client request. It is expected that this feature will be used within ProcessingStages [I-D.goldstein-processing-stages-metadata] to allow a client request to be marked to bypass caching (see Informative Examples (Section 4))
- * Type: Boolean
- * Mandatory-to-Specify: No. The default is "False".

Example usage:

```
{
  "generic-metadata-type": "MI.CacheBypassPolicy",
  "generic-metadata-value": {
    "bypass-cache": true
  }
}
```

Figure 7

3.5. MI.ComputedCacheKey

It is typical in advanced CDN configurations to generate cache keys that are dynamically constructed via lightweight processing of various properties of the HTTP request and/or response. As an example, an upstream source might specify a cache key as a value returned in a specific HTTP response header.

MIComputedCacheKey is a new GenericMetadata object that allows the specification of a cache key using the Metadata Expression Language (MEL) defined in [I-D.power-metadata-expression-language]. Typical use cases would involve constructing a cache key from one or more elements of the HTTP request. In cases where both the MIComputedCacheKey and the Cache object are applied, the MIComputedCacheKey MUST take precedence.

MIComputedCacheKey is, by default, allowed within any of the processing stages defined in [I-D.goldstein-processing-stages-metadata]. It should be noted, however, that a dCDN MAY only allow cache keys to be altered at certain processing stages (such as the clientRequestStage) but not at other stages (such as the originResponse or clientResponseStage). A dCDN MAY use FCI.MetadataExtended [SVTA2041] to advertise such restricted usage contexts.

Property: expression

- * Description: The expression that specifies how the cache key is to be constructed.
- * Type: String. An expression using the Metadata Expression Language (MEL) [I-D.power-metadata-expression-language] to dynamically construct the cache key from elements of the HTTP request and/or response.
- * Mandatory-to-Specify: Yes

In the following example, a custom request header is used as the cache key instead of the Uniform Resource Identifier (URI) path:

```
{
  "generic-metadata-type": "MI.ComputedCacheKey",
  "generic-metadata-value": {
    "expression": "req.h.X-Cache-Key"
  }
}
```

Figure 8

4. Informative Examples

In the following example, cache policies are set in the context of the Processing Stages Model (see the Processing Stages Metadata Specification [I-D.goldstein-processing-stages-metadata]). If the HTTP status code received from the upstream source is a 200, cache expiration is set to 300 seconds if no caching directives were set from the uCDN (unforced). If the HTTP status code received from the upstream source is a 503 or 504, the internal CDN caching policy is forced to 5 seconds and the external downstream policy is forced to "no-cache".

```

{
  "generic-metadata-type": "MI.OriginResponseStage",
  "generic-metadata-value": {
    "match-groups": [
      {
        "if-rule": {
          "match": {
            "expression": "resp.status == 200"
          },
          "stage-metadata": {
            "generic-metadata": [
              {
                "generic-metadata-type": "MI.CachePolicy",
                "generic-metadata-value": {
                  "internal": 300,
                  "external": 300
                }
              }
            ]
          }
        },
        "else-if-rules": [
          {
            "match": {
              "expression": "resp.status == 503 or resp.status == 504"
            },
            "stage-metadata": {
              "generic-metadata": [
                {
                  "generic-metadata-type": "MI.CachePolicy",
                  "generic-metadata-value": {
                    "internal": 5,
                    "external": "no-cache",
                    "force-internal": true,
                    "force-external": true
                  }
                }
              ]
            }
          }
        ]
      }
    ]
  }
}

```

Figure 9

In the next example, an `MI.CacheBypassPolicy` is invoked when the HTTP request header named `"cdn-bypass"` is true. Processing Stages [I-D.goldstein-processing-stages-metadata] and the Metadata Expression Language (MEL) [I-D.power-metadata-expression-language] are used to inspect the HTTP headers as the request is received from the client.

```
{
  "generic-metadata-type": "MI.ProcessingStages",
  "generic-metadata-value": {
    "client-request": [
      {
        "match": {
          "expression": "req.h.cdn-bypass == 'true'"
        },
        "stage-metadata": {
          "generic-metadata": [
            {
              "generic-metadata-type": "MI.CacheBypassPolicy",
              "generic-metadata-value": {
                "bypass-cache": true
              }
            }
          ]
        }
      ]
    ]
  }
}
```

Figure 10

5. Security Considerations

The FCI and MI objects defined in this document are transferred via the interfaces defined in CDNI [RFC8006] which describes how to secure these interfaces by protecting integrity and confidentiality while ensuring the authenticity of the dCDN and uCDN.

6. IANA Considerations

6.1. CDNI Payload Types

This document requests the registration of the following entries under the "CDNI Payload Types" registry hosted by IANA:

Payload Type	Specification
MI.CachePolicy	RFCthis
MI.NegativeCachePolicy	RFCthis
MI.StaleContentCachePolicy	RFCthis
MI.CacheBypassPolicy	RFCthis
MI.ComputedCacheKey	RFCthis

Table 1: CDNI Payload Types

7. Acknowledgements

The authors would like to express their gratitude to the members of the Streaming Video Technology Alliance [SVTA] Open Caching Working Group for their contributions and guidance.

Particularly the following people contribute in one or other way to the content of this draft:

- * Guillaume Bichot (Broadpeak)
- * Christoph Neumann (Broadpeak)
- * Pankaj Chaudhari (Disney Streaming)
- * Robert Colantuoni (Disney Streaming)
- * Rajeev RK (picoNETS)
- * Yoav Gressel (Qwilt)
- * Noam Peleg (Qwilt)
- * Arnon Warshavsky (Qwilt)
- * Eric Klein (Sirius XM)
- * Alfonso Siloniz (Telefonica)
- * Ben Rosenblum (Vecima)

8. Normative References

- [I-D.power-metadata-expression-language]
Power, W., Goldstein, G., and A. Warshavsky, "CDNI Metadata Expression Language", Work in Progress, Internet-Draft, draft-power-metadata-expression-language-03, 25 February 2025, <<https://datatracker.ietf.org/doc/html/draft-power-metadata-expression-language-03>>.
- [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>>.
- [RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <<https://www.rfc-editor.org/info/rfc8006>>.
- [RFC9110] 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>>.
- [RFC9111] 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>>.

9. Informative References

- [I-D.goldstein-processing-stages-metadata]
Goldstein, G., Power, W., and A. Warshavsky, "CDNI Processing Stages Metadata", Work in Progress, Internet-Draft, draft-goldstein-processing-stages-metadata-03, 25 February 2025, <<https://datatracker.ietf.org/doc/html/draft-goldstein-processing-stages-metadata-03>>.
- [SVTA] SVTA, "Streaming Video Technology Alliance Home Page", <<https://www.svta.org>>.
- [SVTA2041] SVTA, "Metadata Capabilities", <<https://svta.org/documents/SVTA2041>>.

Authors' Addresses

Will Power
Lumen Technologies
United States of America
Email: wrpower@gmail.com

Glenn Goldstein
Lumen Technologies
United States of America
Email: glennng1215@gmail.com