Content Delivery Networks Interconnection
Internet-Draft
Intended status: Standards Track
Expires: 26 February 2026

G. Goldstein
W. Power
Lumen Technologies
A. Warshavsky
Qwilt

25 August 2025

CDNI Processing Stages Metadata
draft-goldstein-processing-stages-metadata-04

Abstract

This document specifies a set of objects extending the Content Delivery Network Interconnection (CDNI) metadata model to allow for metadata to be applied conditionally and at various points in a dCDNs processing of requests. The concept of Processing Stages are introduced, where each stage in a CDN’s processing pipeline presents an opportunity to examine requests and responses and make alterations as needed. Metadata, such as caching rules, can be applied conditionally (based on aspects of an HTTP request header), and HTTP responses from a source can be altered dynamically (such as adding or dropping an HTTP header). This standard leverages the expression language documented in the Metadata Expression Language (MEL) Specification.

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 26 February 2026.

Copyright Notice

Table of Contents

1. Introduction

It is typical in CDN configurations to define matching rules and metadata to be applied at specific stages in the HTTP request processing pipeline. For example, it may be required to append a host header prior to forwarding a request to an origin, or modify the response returned from an origin prior to storing in the cache.

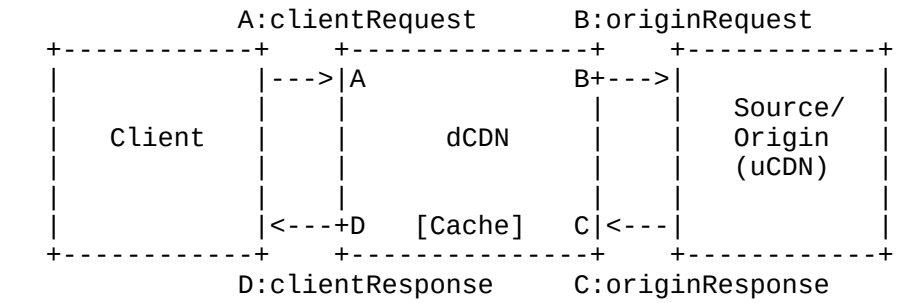


Figure 1: Processing Stages Context

The four processing stages are:

* clientRequest - Rules run on the inbound client HTTP request prior to further processing.

* originRequest - Rules run prior to making an HTTP request to the origin upon a cache miss.

* originResponse - Rules run after an HTTP response is received from the origin and before being placed in the cache or forwarded to the client.

* clientResponse - Rules run prior to sending the HTTP response to the client. If the response is from the cache, rules are applied to the response retrieved from the cache prior to sending to the client.

Note that all references HTTP requests, responses, and headers use the semantics defined in [RFC9110]

Requirements of the processing stages are:

* Header Matching - While CDNI metadata defines some basic matching rules for host names and pattern patching on paths, CDN and Open Caching use cases often require matching on specific fields in a Hypertext Transfer Protocol (HTTP) request and response headers to set metadata. A typical example may be matching on a user agent string to set access controls or matching on a Multipurpose Internet Mail Extensions-type (MIME-type) header to set caching rules. A rich expression-matching syntax that allows matching on any combination of host, path, and header values covers most typical use cases.

* Expression Matching - Header matching alone is not always sufficient for identifying a set of requests or responses that require specific metadata. CDN and Open Caching systems often require a rich set of matching rules, with full regular expressions and Boolean combinations of matching parameters for host, path, and header elements of a request. In typical CDN implementations, this capability is provided by a rich expression language that can be embedded in the metadata configurations.

* URI Modifications - In processing HTTP requests, modifications to the request Uniform Resource Identifier (URI) are often required for uses such as collapsing multiple paths to a common cache key or normalizing file extension naming conventions before making a request to the origin. In cases where the modified URI needs to be constructed dynamically, an expression language is provided that allows elements of requests and responses to be concatenated with string literals.

* Header Modifications - In processing HTTP requests, it is often required to modify HTTP request or response headers at one of the processing stages, requiring CDNI metadata to have the capability to update any field in an HTTP request or response header. It should be noted that certain HTTP headers (such as Set-Cookie) have multiple occurrences in a request or response, thereby requiring that we allow for add and replace designations for header modification. In cases where a header value needs to be constructed dynamically, an expression language is provided that allows elements of requests and responses to be concatenated with string literals. The following capabilities SHOULD be supported at each processing stage:

  - Add Request Header Field - Add a header name/value to the request, along with any headers of the same name that may already be present.

  - Replace Request Header Field - Add a header name/value to the request, replacing any headers of the same name that may already be present.

  - Delete Request Header Field - Delete all occurrences of the named header from the request.

  - Add Response Header Field - Add a header name/value to the response, along with any headers of the same name that may already be present.

- Replace Response Header Field - Add a header name/value to the response, replacing any headers of the same name that may already be present.

- Delete Response Header Field - Delete all occurrences of the named header from the response.

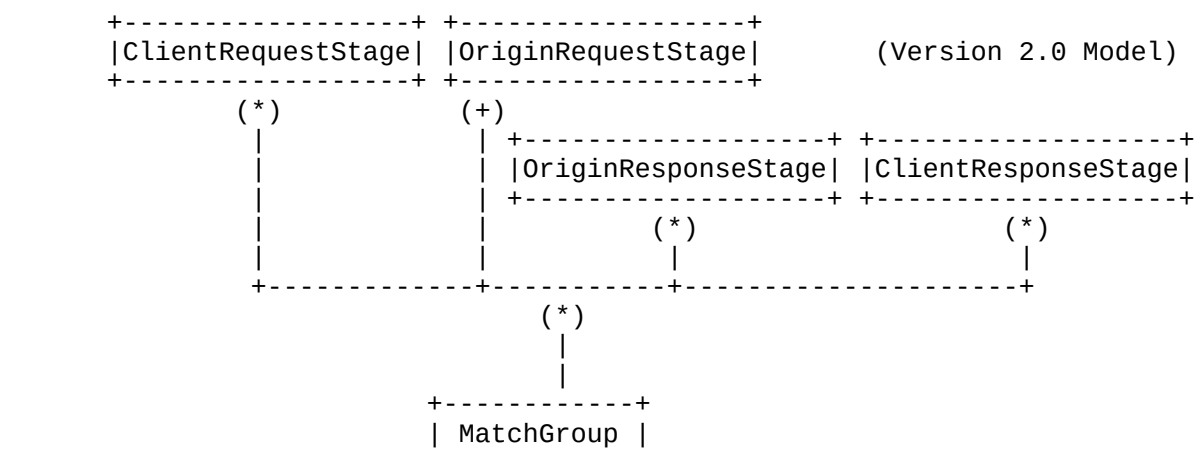
* Synthetic Responses - It is quite common in CDN configurations to specify a synthetic response be generated based on inspection of aspects of the original request or the origin response. The synthetic response capability allows for the specification of a set of response headers, a status code, and a response body. In cases where a header value or the synthetic response body needs to be constructed dynamically, an expression language is provided that allows elements of requests and responses to be concatenated with string literals.

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. Processing Stages Object Model

The four processing stages and their associated match criteria and metadata are defined as a new set of GenericMetadata objects, all under the structure of top level processing stages objects. Each stage contains an OPTIONAL matching expression (determining if the stage rules should be applied), along with a set of rules (i.e., metadata) to be applied upon a match. Note that the expression match is optional; if none is provided, the stage rules are always applied.



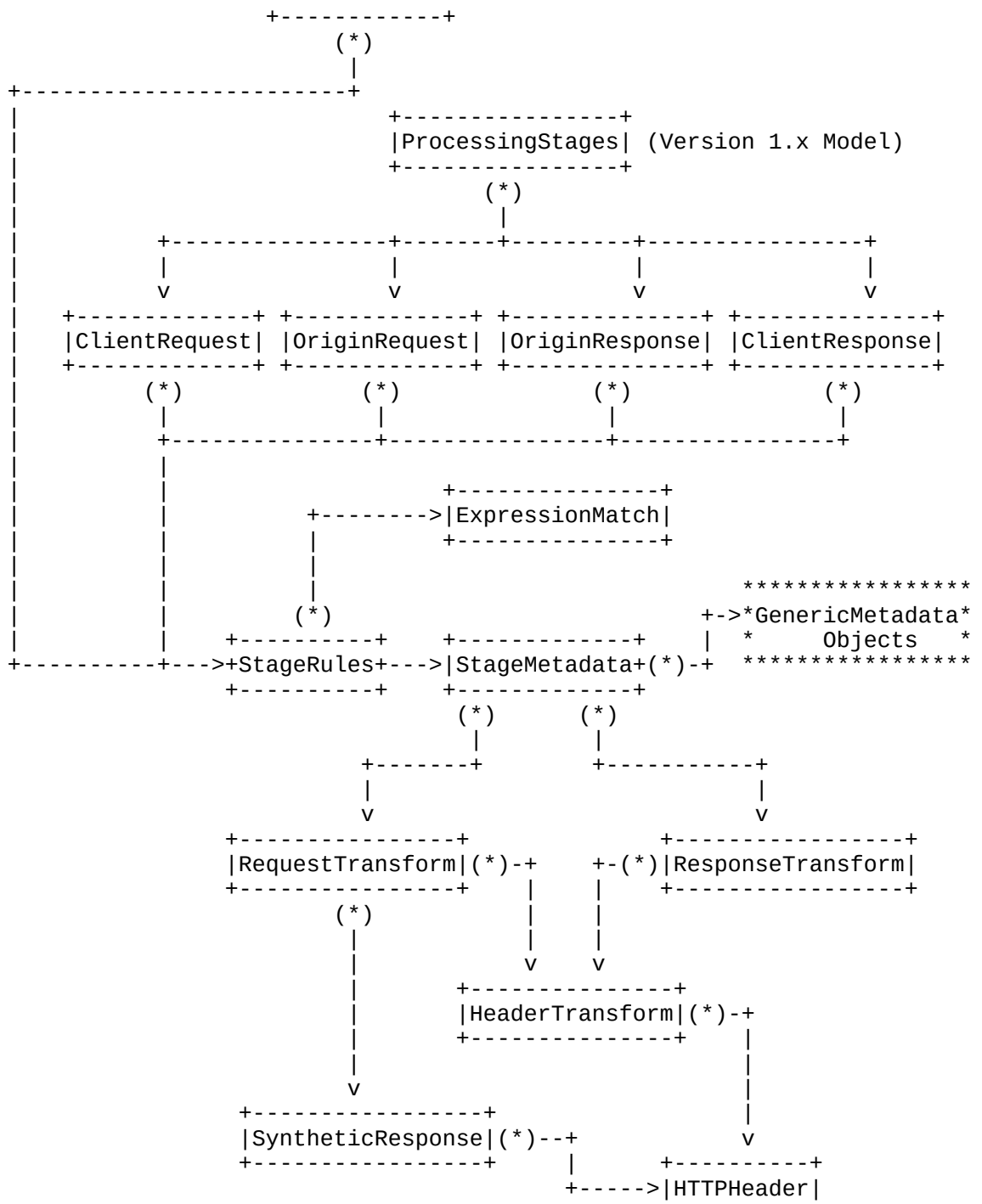

ProcessingStages
(Version 1.x Model)
ClientRequest
OriginRequest
OriginResponse
ClientResponse
ExpressionMatch
GenericMetadata
Objects
StageRules
StageMetadata
RequestTransform
ResponseTransform
HeaderTransform
SyntheticResponse
HTTPHeader

Figure 2: Processing Stages Model

Each of the four processing stages is represented by arrays of MI.StageRules objects, with each MI.StageRules object defining criteria along with metadata that MUST be applied if the match applies to "True". All of the MI.StageRules objects in each stage's array are evaluated and processed in order, with processing only terminated if the stage metadata generates a synthetic response or denies access via an access control list (ACL) restriction. See the MI.StageRules (Section 3.2) section for more information on terminating and non-terminating objects and the order of execution.

For backward compatibility with earlier versions of this specification, two options are provided for structuring processing stages metadata:

* A single MI.ProcessingStages object, where all four processing stages are combined into a single structure.

* Distinct MI objects for each processing stage, allowing configuration metadata to be managed in more atomic units. These new objects introduce MI.MatchGroup to provide a convenient if/else structure for conditionally applying metadata.

3.1. MI.ProcessingStages

MI.ProcessingStages is a GenericMetadata object that describes the matching rules, metadata, and transformations to be applied at specific stages in the request processing pipeline.

If a configuration requires if/else logic (halting processing once an expression match is achieved), it is recommended using the MI.ClientRequestStage, MI.ClientResponseStage, MI.OriginRequestStage, or MI.OriginResponseStage objects with MI.MatchGroups (see below).

Property: client-request

* Description: Allows for the specification of conditional metadata to be applied at the client request processing stage. The MI.StageRules in the array are evaluated in order.

* Type: Array of MI.StageRules objects

* Mandatory-to-Specify: No

Property: origin-request

* Description: Allows for the specification of conditional metadata to be applied at the origin request processing stage. The MI.StageRules in the array are evaluated in order.

* Type: Array of MI.StageRules objects

* Mandatory-to-Specify: No

Property: origin-response

* Description: Allows for the specification of conditional metadata to be applied at the origin response processing stage. The MI.StageRules in the array are evaluated in order.

* Type: Array of MI.StageRules objects

* Mandatory-to-Specify: No

Property: client-response

* Description: Allows for the specification of conditional metadata to be applied at the client response processing stage. The MI.StageRules in the array are evaluated in order.

* Type: Array of MI.StageRules objects

* Mandatory-to-Specify: No

The following example specifies all four processing stages. In this example, the client-request stage has two MI.StageRules, appling one set of metadata if ExpressionMatch1 evaluates to "True" and applying another set of metadata if ExpressionMatch2 evaluates to "True".

```
{
  "generic-metadata-type": "MI.ProcessingStages",
  "generic-metadata-value": {
    "client-request" : [
      {
        "match" :         <ExpressionMatch1 for conditional metadata>
        "stage-metadata" : <StageMetadata1 for clientRequest stage>,
      },
      {
        "match" :         <ExpressionMatch2 for conditional metadata>
        "stage-metadata" : <StageMetadata2 for clientRequest stage>,
      }
    ],
    "origin-request" : [{
      "match" :           <ExpressionMatch for conditional metadata>
      "stage-metadata" :   <StageMetadata for originRequest stage>,
    }],
    "origin-response" : [{
      "match" :           <ExpressionMatch for conditional metadata>
      "stage-metadata" :   <StageMetadata for originResponse stage>,
    }],
    "client-response" : [{
      "match" :           <ExpressionMatch for conditional metadata>
      "stage-metadata" :   <StageMetadata for clientResponse stage>,
    }]
  }
}
```

Figure 3

3.1.1. MI.ClientRequestStage

MI.ClientRequestStage is a GenericMetadata object that describes the matching rules, metadata, and transformations to be applied by the dCDN upon receiving an inbound HTTP request from a client prior to further processing.

Property: match-groups

* Description: An array of if/else match groups that are processed in order, allowing for the specification of conditional metadata to be applied at this processing stage. Note that all MI.MatchGroup objects in the array are evaluated and processed in order, with processing only halted if the stage metadata generates a synthetic response or denies access via an ACL restriction.

* Type: Array of MI.MatchGroup objects

* Mandatory-to-Specify: Yes

3.1.2. MI.OriginRequestStage

MI.OriginRequestStage is a GenericMetadata object that describes the matching rules, metadata, and transformations to be applied by the dCDN prior to making an HTTP request to the origin upon a cache miss.

Property: match-groups

* Description: An array of if/else match groups that are processed in order, allowing for the specification of conditional metadata to be applied at this processing stage. Note that all MI.MatchGroup objects in the array are evaluated and processed in order, with processing only halted if the stage metadata generates a synthetic response or denies access via an ACL restriction.

* Type: Array of MI.MatchGroup objects

* Mandatory-to-Specify: Yes

3.1.3. MI.OriginResponseStage

MI.OriginResponseStage is a GenericMetadata object that describes the matching rules, metadata, and transformations to be applied by the dCDN after an HTTP response is received from the origin and before being placed in the cache or forwarded to the client.

Property: match-groups

* Description: An array of if/else match groups that are processed in order, allowing for the specification of conditional metadata to be applied at this processing stage. Note that all MI.MatchGroup objects in the array are evaluated and processed in order, with processing only halted if the stage metadata generates a synthetic response or denies access via an ACL restriction.

* Type: Array of MI.MatchGroup objects

* Mandatory-to-Specify: Yes

The following example illustrates an MI.OriginResponseStage with two match groups. The first match group uses a simple "if/else" construct, while the second match group only contains an "if" construct (with no "else").

```
{
  "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": "5",
                  "external": "no-cache",
                  "force-internal": false,
                  "force-external": false
                }
              }
            ]
          }
        },
        "else-if-rules": [
          {}
        ]
      },
      {
        "if-rule": {
          "match": {
            "expression": "$req.h.x-test-ignore-cache == 'edge'
                     OR $req.h.x-test-ignore-cache == 'parent'"
          },
          "stage-metadata": {
            "generic-metadata": [
              {
                "generic-metadata-type": "MI.CacheBypassPolicy",
                "generic-metadata-value": {
                  "bypass-cache": true
                }
              }
            ]
          }
        }
      }
    ]
  }
}
```

Figure 4

3.1.4. MI.ClientResponseStage

MI.OriginResponseStage is a GenericMetadata object that describes the matching rules, metadata, and transformations to be applied by the dCDN prior to sending the HTTP response to the client. If the response is from the cache, rules are applied to the response retrieved from the cache prior to sending to the client.

Property: match-groups

* Description: An array of if/else match groups that are processed in order, allowing for the specification of conditional metadata to be applied at this processing stage. Note that all MI.MatchGroup objects in the array are evaluated and processed in order, with processing only halted if the stage metadata generates a synthetic response or denies access via an ACL restriction.

* Type: Array of MI.MatchGroup objects

* Mandatory-to-Specify: Yes

3.1.5. MI.MatchGroup

MI.MatchGroup is a subobject of MI.ClientRequestStage, MI.ClientResponseStage, MI.OriginRequestStage, and MI.OriginResposeStage, allowing for a structured set of if/else rules to be applied based on an OPTIONAL match of an expression. At a high-level, the following structure is supported:

```
if (expression match A)
    <Apply a set of metadata>
else if (expression match B)
    <Apply an alternate set of metadata>
else if (expression match C)
    <Apply an alternate set of metadata>
else
    <Apply a default set of metadata>
```

Figure 5

Property: if-rule

* Description: An MI.StageRules object that is always processed. If it contains a match expression that evaluates to "True" (or is an unconditional rule with no match expression), the metadata defined within the stage rule is applied and any else-if-rules are ignored.

* Type: MI.StageRules object

* Mandatory-to-Specify: Yes

Property: else-if-rules

* Description: An array of MI.StageRules objects that are processed in order whenever the if-rule does not terminate processing. If a stage rule expression match evaluates to "True" (or is an unconditional rule with no match expression), the metadata defined within the stage rule is applied and all subsequent entries in the else-if-rules are ignored. Processing is halted if the stage metadata generates a synthetic response or denies access via an ACL restriction.

* Type: Array of MI.StageRules objects

* Mandatory-to-Specify: No

The following MI.MatchGroup example illustrates a simple if/else structure for applying metadata at the origin response stage based on the HTTP status code:

```
{
  "generic-metadata-type": "MI.MatchGroup",
  "generic-metadata-value": {
    "if-rule": {
      "match": {
        "expression": "resp.status == 200"
      },
      "stage-metadata": {
        "generic-metadata": [
          {}
        ]
      }
    },
    "else-if-rules": [
      {
        "match": {
          "expression": "resp.status == 304"
        },
        "stage-metadata": {
          "generic-metadata": [
            {}
          ]
        }
      },
      {
        "stage-metadata": {
          "generic-metadata": [
            {}
          ]
        }
      }
    ]
  }
}
```

Figure 6

3.2. MI.StageRules

An MI.StageRules object is used within the context of ProcessingStages to define elements in a list of match rules and stage-specific metadata and transformations that MUST be applied conditionally on a rich expression match.

Processing is terminated at any StageRule and enclosing MatchGroup that either generates a synthetic response (via MI.SyntheticResponse) or denies access (via LocationACL,LocationACLExtended,TimeWindowACL,or

TimeWindowACLExtended). These MI objects that terminate processing are henceforth referred to as terminating objects, and the logical order of overriding a terminating object differs from that of a non-terminating object, as illustrated in the diagram below.

The flow on the right-side of the diagram illustrates the order of execution for non-terminating objects such as MI.Cache or MI.Source. The object defined at the host or path level is selected, and may be overridden with any other instance of that same generic metadata object that is processed as a result of being in a MI.StageRules with a successful match criteria. The exception to the override rule is recurring objects such as MI.SetVariable [I-D.power-metadata-expression-language] where all instances of the object in a stage with a matching expression are executed without override. This is necessary to allow situations where different match criteria may be used to set different user-defined variables.

The flows on the left-side of the diagram illustrate the order of execution for terminating objects that may either allow or deny access. If a terminating object evaluated in a matched rule of a processing stage is selected for execution and results in an "allow" policy, subsequent objects of the same GenericMetadata type will not be executed and an implicit "allow" will be assumed. In all cases, the order of execution is not affected by the position of the ProcessingStages objects in a GenericMetadata object array.

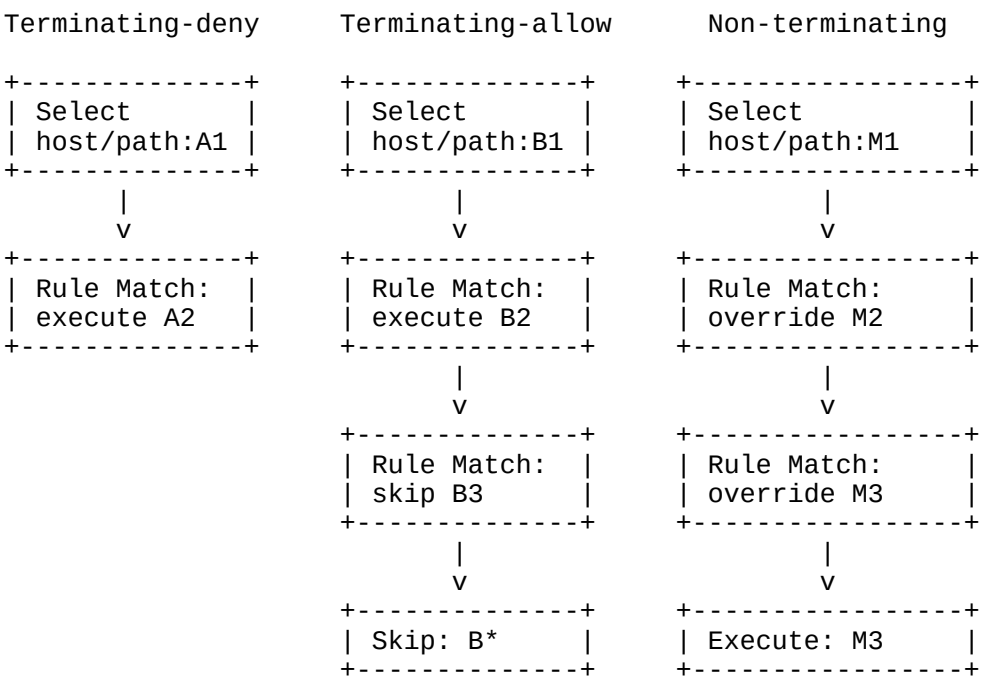


Figure 7: Processing Stages order of execution for terminating objects

In the case of private features [SVTA2038], the classification of the object as either terminating, non-terminating, or recurring is implementation-specific. As an example, a private feature could be implemented to invoke custom scripts that may or may not terminate execution or require override behavior.

Property: match

* Description: An ExpressionMatch object encapsulating a rich expression using the CDNI Metadata Expression Language to evaluate aspects of the HTTP request and/or response. The stage-metadata rules are only applied if the match evaluates to "True" or if no match expression is provided.

* Type: ExpressionMatch object

* Mandatory-to-Specify: No. The stage-metadata rules are considered unconditional and are always applied if no match expression is provided. This would be the case when stage-metadata should be applied unconditionally within the context of the higher-level host and path matches.

Property: stage-metadata

* Description: Specifies the StageMetadata to be applied at the processing stage if the match expression evaluates to "True" or is not present.

* Type: StageMetadata object

* Mandatory-to-Specify: Yes

The following is an example of StageRules that are applied just after responses are received from the origin. In this example, receipt of a response status code of 304 from the origin indicates that CachePolicy metadata SHOULD be applied (as specified via an external HREF), and that response headers SHOULD be modified (X-custom-response-header added and ETag deleted).

```
{
  "generic-metadata-type": "MI.StageRules",
  "generic-metadata-value": {
    "match": {
      "expression": "resp.status == 304"
    },
    "stage-metadata": {
      "generic-metadata": [
        {
          "type": "MI.CachePolicy",
          "href":
             "https://metadata.ucdn.example/origin_response_cache"
        }
      ],
      "response-transform": {
        "header-transform": {
          "add": [
            {
              "name": "X-custom-response-header",
              "value": "header-value"
            }
          ],
          "delete": [
            "ETag"
          ]
        }
      }
    }
  }
}
```

Figure 8

3.2.1. MI.ExpressionMatch

The MI.ExpressionMatch object contains the rich expression that MUST evaluate to "True" for the StageMetadata to be applied for the specific StageRules. Defining expressions as standalone objects allows for sets of match expressions to be reused via metadata reference linking.

Property: expression

* Description: A rich expression using MEL to evaluate aspects of the HTTP request and/or response. See documentation on MEL [I-D.power-metadata-expression-language] for details on the expression of matching variables and syntax.

* Type: String, adhering to the Metadata Expression Language (MEL) syntax.

* Mandatory-to-Specify: Yes

The following is an example of ExpressionMatch on the referrer and user agent request headers:

```
{
  "generic-metadata-type": "MI.MatchExpression",
  "generic-metadata-value": {
    "expression": "req.h.user-agent *= '*Safari*'
          and req.h.referrer == 'www.x.com'"
  }
}
```

Figure 9

3.2.2. MI.StageMetadata

The MI.StageMetadata object contains GenericMetadata and HTTP request/response transformations that MUST be applied for a StageRules match. The following table defines the processing stages where request and response transformations are possible:

Stage	request-transform	response-transform
clientRequest	yes	yes
originRequest	yes	yes
originResponse	no	yes
clientResponse	no	yes

Table 1

Note that for the stages where both request and response transformations are allowed, it is possible to specify both. This may be the case if, for example, the request URI needs alteration for cache key generation and the response headers need to be manipulated.

Property: generic-metadata

* Description: Specifies the set of GenericMetadata to be applied for a StageRules match. A typical use case would be the application of a CachePolicy or TimeWindowACL conditionally on matching HTTP headers. It should be noted that not all GenericMetadata types are valid here. Specifically, one MUST NOT include any of the ProcessingStages objects in this array, as that would create nested processing stage scenarios that add unnecessary complexity. Furthermore, there MAY be GenericMetadata types defined across various specifications that SHOULD NOT be used within a processing stages context. These context restrictions can be expressed using FCI.MetadataExtended, as documented in the Metadata Capabilities Specification [SVTA2041]

Note that the GenericMetadata objects in this array must adhere to the guidelines documented in Section 3.3(Metadata Inheritance and Override) of [RFC8006] Specifically, the array MUST NOT contain multiple GenericMetadata objects of the same type, and any metadata specified in this array overrides all GenericMetadata objects of the same type previously defined by any parent object in the HostMetadata or PathMetadata tree.

* Type: Array of GenericMetadata, applied in order.

* Mandatory-to-Specify: No. The GenericMetadata property would not be needed when StageMetadata is used to only specify request or response transformations, such as modifications of HTTP headers.

Property: request-transform

* Description: Specifies a transformation to be applied to the HTTP request for a StageRules match. The transformation can be the modification of any request header and/or the modification of the URI. Modifications are applied such that downstream processing stages receive the modified HTTP request as their input. Support for this capability is OPTIONAL and can be advertised using FCI.MetadataExtended, as documented in the Metadata Capabilities Specification [SVTA2041]

* Type: RequestTransform object

* Mandatory-to-Specify: No

Property: response-transform

* Description: Specifies a transformation to be applied to the HTTP response for a StageRules match. The transformation can be the modification of any response header, HTTP response status code, or the generation of a synthetic response Modifications are applied such that downstream processing stages receive the modified HTTP response as their input. Support for this capability is OPTIONAL and can be advertised using FCI.MetadataExtended, as documented in the Metadata Capabilities Specification [SVTA2041]

* Type: ResponseTransform object

* Mandatory-to-Specify: No

The following is an example of structure of a StageMetadata object:

```
{
  "generic-metadata" :  [{
    < Optional list of generic metadata to apply at this stage >
  }],
  "request-transform" : {
    "header-transform" : { <list request headers to modify> },
    "uri" : < URI rewrite, either static or dynamically constructed>
  }
  "response-transform" : {
    "header-transform" : { <list response headers to modify> },
    "response-status" :    <static or dynamically constructed status>
  }
}
```

Figure 10

3.2.2.1. MI.RequestTransform

The MI.RequestTransform object contains metadata for transforming the HTTP request for a specific StageRules object. The transformation can be the modification of any request header and/or the modification of the URI. Modifications are applied such that downstream processing stages receive the modified HTTP request as their input.

Property: header-transform

* Description: A HeaderTransform object specifying HTTP request headers to add, replace, or delete.

* Type: HeaderTransform object

* Mandatory-to-Specify: No

Property: uri

* Description: A replacement value for the HTTP request.

* Type: String. Either a literal (static string) or an expression using MEL [I-D.power-metadata-expression-language] to dynamically construct a URI value from elements of the HTTP request and/or response.

* Mandatory-to-Specify: No

Property: uri-is-expression

* Description: A flag to signal whether the URI is a static string literal or a MEL expression that needs to be dynamically evaluated.

* Type: Boolean

* Mandatory-to-Specify: No. The default is "False", indicating that the URI is a string literal and does not need to be evaluated.

The following is an example of a RequestTransform object illustrating a dynamically constructed URI rewrite:

```
{
  "generic-metadata-type": "MI.RequestTransform",
  "generic-metadata-value": {
    "header-transform": {},
    "uri": "req.uri.path",
    "uri-is-expression": true
  }
}
```

Figure 11

3.2.2.2. MI.ResponseTransform

The MI.ResponseTransform object contains metadata for transforming the HTTP response for a StageRules match. The transformation can be the modification of any response header, HTTP response status code, or the generation of a synthetic response. Modifications are applied such that downstream processing stages receive the modified HTTP response as their input.

Property: header-transform

* Description: A HeaderTransform object specifying HTTP response headers to add, replace, or delete.

* Type: HeaderTransform object

* Mandatory-to-Specify: No

Property: response-status

* Description: A replacement value for the HTTP response status code.

* Type: Integer. Either a static integer or an expression using MEL [I-D.power-metadata-expression-language] that evaluates to an integer to dynamically generate an HTTP status code based on elements of the HTTP request and/or response. Expressions that do not evaluate to an integer shall be considered invalid and result in no override of origin-provided response status.

* Mandatory-to-Specify: No

Property: status-is-expression

* Description: A flag to signal whether the response-status is a static integer or a MEL expression that needs to be dynamically evaluated to generate an HTTP response status code.

* Type: Boolean

* Mandatory-to-Specify: No. The default is "False", indicating that the response-status is a static integer and does not need to be evaluated.

Property: synthetic

* Description: The specification of a complete replacement of any HTTP response that may have been generated in an earlier processing stage with a synthetic response. Use of this property to specify a synthetic response would override any response transformations or status codes specified by other properties.

* Type: SyntheticResponse object

* Mandatory-to-Specify: No

The following is an example of a ResponseTransform object, illustrating a dynamically constructed header value that uses the metadata expression language [I-D.power-metadata-expression-language] to concatenate the user agent and host header, and forces a 403 HTTP response status code:

```
{
  "generic-metadata-type": "MI.ResponseTransform",
  "generic-metadata-value": {
    "header-transform": {
      "add": [
        {
          "name": "X-custom-response-header",
          "value": "req.h.user-agent . '-' . req.h.host",
          "value-is-expressions": true
        }
      ]
    },
    "response-status": "403"
  }
}
```

Figure 12

3.2.2.3. MI.SyntheticResponse

The MI.SyntheticResponse object allows for the specification of a synthetic response to be generated in response to the HTTP request being processed. The synthetic response can contain a set of response headers, a status code, and a response body, and is a complete replacement for any HTTP response elements generated in an earlier processing stage. Note that MI.SyntheticResponse is handled as a terminating object (see MI.StageRules (Section 3.2)) in that processing is terminated upon execution.

A dynamically generated Content-Length HTTP response header is generated based on the length of the generated response body.

Property: headers

* Description: An array of HTTP header objects that specifies the full set of headers to be applied to the synthetic response.

* Type: Array of HTTP header objects

* Mandatory-to-Specify: No, although it would be unusual to not specify minimal standard response headers, such as Content-Type.

Property: response-status

* Description: The HTTP response status code.

* Type: Either a static integer or a string containing an expression that evaluates to an integer using MEL [I-D.power-metadata-expression-language] to dynamically generate an HTTP status code based on elements of the upstream HTTP request and/or response. Expressions that do not evaluate to an integer shall be considered invalid and result in a 500 status for the synthetic response.

* Mandatory-to-Specify: Yes

Property: status-is-expression

* Description: The flag to signal whether the response-status is a static integer or a MEL expression [I-D.power-metadata-expression-language] that needs to be dynamically evaluated to generate an HTTP response status code.

* Type: Boolean

* Mandatory-to-Specify: No. The default is "False", indicating that the response-status is a static integer and does not need to be evaluated.

Property: body

* Description: The body for the synthetic HTTP response. The response body can either be static or dynamically constructed from a rich expression.

* Type: String. Either a literal (static string) or an expression using MEL [I-D.power-metadata-expression-language] to dynamically construct a response body from elements of the HTTP request and/or response.

* Mandatory-to-Specify: No. If absent, an empty HTTP response with a zero-value Content-Length header is generated.

Property: body-is-expression

* Description: A flag to signal whether the synthetic response body is a static string literal or a MEL expression [I-D.power-metadata-expression-language] that needs to be dynamically evaluated.

* Type: Boolean

* Mandatory-to-Specify: No. The default is "False", indicating that the body is a string literal and does not need to be evaluated.

The following is an example of a SyntheticResponse object illustrating a dynamically constructed response body that uses the expression language to combine the request URI with static text and forces a 405 HTTP response status code:

```
{
  "generic-metadata-type": "MI.SyntheticResponse",
  "generic-metadata-value": {
    "headers": [
      {
        "name": "content-type",
        "value": "text/plain"
      },
      {
        "name": "X-custom-response-header",
        "value": "some static value"
      }
    ],
    "response-status": 405,
    "response-body":
      "'Sorry, Access to resource ' . req.uri . ' not allowed'",
    "body-is-expression": true
  }
}
```

Figure 13

3.2.2.4. MI.HeaderTransform

The MI.HeaderTransform object specifies how HTTP requests and response headers can be transformed. Headers can be deleted, replaced, or added to an HTTP request or response, with the processing occurring in this order: DELETE, REPLACE, ADD.

Property: delete

* Description: A list of names of HTTP headers that MUST be deleted from the HTTP request or response. If a named header appears multiple times, all occurrences are deleted.

* Type: Array of strings, with each string naming an HTTP header to delete

* Mandatory-to-Specify: No

Property: replace

* Description: A list of HTTP headers (name/value pairs) that MUST be added to the HTTP request or response, replacing any previous headers with the same name.

* Type: Array of HTTPHeader objects containing header name/value pairs

* Mandatory-to-Specify: No

Property: add

* Description: A list of HTTP headers (name/value pairs) that MUST be added to the HTTP request or response. Note that any existing headers in the request or response with the same names of those added are not affected, resulting in multiple headers with the same name.

* Type: Array of HTTPHeader objects containing header name/value pairs

* Mandatory-to-Specify: No

The following is an example of a HeaderTransform object illustrating the addition of two customer headers, the replacement of any previously provided Accept-Encoding header, and the removal of any previously provided Authorization or Accept-Language headers:

```
{
  "generic-metadata-type": "MI.HeaderTransform",
  "generic-metadata-value": {
    "add": [
      {
        "name": "X-custom-header1",
        "value": "header-value 1"
      },
      {
        "name": "X-custom-header2",
        "value": "header-value 2"
      }
    ],
    "replace": [
      {
        "name": "Accept-Encoding",
        "value": "gzip,deflate,br"
      }
    ],
    "delete": [
      "Authorization",
      "Accept-Language"
    ]
  }
}
```

Figure 14

3.2.2.5. MI.HTTPHeader

The MIHTTPHeader object contains a name/value pair for an HTTP header to add or replace in a request or response. The metadata expression language (MEL) [I-D.power-metadata-expression-language] can be used to dynamically generate response values.

Property: name

* Description: The name of the HTTP header.

* Type: String

* Mandatory-to-Specify: Yes

Property: value

* Description: The new value of the named HTTP header.

* Type: String. Either a static string or an expression using the MEL expression language [I-D.power-metadata-expression-language] to dynamically construct a header value from elements of the HTTP request and/or response.

* Mandatory-to-Specify: Yes

Property: value-is-expression

* Description: A flag to signal whether the value is a static string literal or a MEL expression [I-D.power-metadata-expression-language] that needs to be dynamically evaluated.

* Type: Boolean

* Mandatory-to-Specify: No. The default is "False", indicating that the value is a string literal and does not need to be evaluated.

The following is an example of an HTTPHeader illustrating a dynamically constructed header value that equals the session parameter from the query string:

```
{
  "generic-metadata-type": "MI.HTTPHeader",
  "generic-metadata-value": {
    "name": "X-custom-response-header",
    "value": "req.uri.query.session",
    "value-is-expression": true
  }
}
```

Figure 15

4. Metadata Expression Language Usage

Throughout the Processing Stages Model, the CDNI Metadata Expression Language (MEL) is used in two ways:

* Match Expressions: Expressions that evaluate to a Boolean value are used to dynamically determine if metadata should be applied based on evaluation of aspects of an inbound HTTP request (matching on a header value, for example).

* Value Expressions: Enable the dynamic construction of a value to be used in scenarios such as constructing a cache key, setting an HTTP response header or status code, rewriting a request URI, or dynamically generating a response body.

Expressions can evaluate to a Boolean value, string, or integer, depending on the use case:

Usage	Description	Evaluation Results
ExpressionMatch.expression	Dynamically determines if StageMetadata should be applied at a specific StageRules.	Boolean. Expressions that do not evaluate to "True" or "False" shall be considered "False".
RequestTransform.uri	Rewrites request URI that will be presented to all downstream stages.	String
ResponseTransform.response-status	Dynamically sets a response status code to replace the status code returned by the origin.	Integer (HTTP status code)
SyntheticResponse.response-status	Dynamically sets a response status code for a synthetically constructed response.	Integer (HTTP status code)

SyntheticResponse.body	Dynamically constructs a response body.	String
HTTPHeader.value	Dynamically constructs a header value.	String

Table 2

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.ProcessingStages	RFCthis
MI.ClientRequestStage	RFCthis
MI.OriginRequestStage	RFCthis
MI.OriginResponseStage	RFCthis
MI.ClientResponseStage	RFCthis
MI.MatchGroup	RFCthis
MI.StageRules	RFCthis
MI.ExpressionMatch	RFCthis
MI.StageMetadata	RFCthis
MI.RequestTransform	RFCthis
MI.ResponseTransform	RFCthis
MI.SyntheticResponse	RFCthis
MI.HeaderTransform	RFCthis
MI.HTTPHeader	RFCthis

Table 3: 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.

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

* Guillaume Bichot - Broadpeak

* Pankaj Chaudhari - Disney Streaming Services

* Yoav Gressel - Qwilt

* Ben Rosenblum - Vecima

* Rajeev RK - picoNETS

* Alfonso Siloniz - Telefonica

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-04, 25 August 2025, <https://datatracker.ietf.org/doc/html/draft-power-metadata-expression-language-04>.

[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>.

9. Informative References

[SVTA] SVTA, "Streaming Video Technology Alliance Home Page", <https://www.svta.org>.

[SVTA2038] SVTA, "Private Features Metadata", <https://svta.org/documents/SVTA2038>.

[SVTA2041] SVTA, "Metadata Capabilities", <https://svta.org/documents/SVTA2041>.

Authors' Addresses

Glenn Goldstein
Lumen Technologies
United States of America
Email: glenng1215@gmail.com

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

Arnon Warshavsky
Qwilt
Israel
Email: arnon@qwilt.com