Workgroup: Network Working Group
Internet-Draft:
draft-goldstein-cdni-metadata-model-
extensions-02
Updates: 8006, 8008 (if approved)
Published: 7 March 2022
Intended Status: Standards Track
Expires: 8 September 2022
Authors: G. Goldstein
Lumen Technologies
W. Power
Lumen Technologies
G. Bichot
Broadpeak
A. Siloniz
Telefonica

CDNI Metadata Model Extensions

Abstract

The Content Delivery Network Interconnection (CDNI) Metadata interface enables interconnected Content Delivery Networks (CDNs) to exchange content distribution metadata in order to enable content acquisition and delivery. To facilitate a wider set of use cases such as Open Caching, this document describes extensions to the CDNI Metadata object model and its associated Capabilities model as documented in "CDNI Metadata" RFC8006 and "CDNI Request Routing: Footprint and Capabilities Semantics" RFC8008 .

This document is a reflection of the content in the Streaming Video Alliance specification titled "SVA Configuration Interface: Part 2 Extensions to CDNI Metadata Object Model".

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 8 September 2022.

Copyright Notice

Table of Contents

1. Introduction and Scope
  1.1. Terminology
  1.2. Requirements Language
2. CDNI Metadata Model Extensions
  2.1. Cache Control Metadata
    2.1.1. MI.CachePolicy
    2.1.2. MI.NegativeCachePolicy
    2.1.3. MI.StaleContentCachePolicy
    2.1.4. MI.CacheBypassPolicy
    2.1.5. MI.ComputedCacheKey
  2.2. Origin Access Metadata
    2.2.1. MI.SourceMetadataExtended
      2.2.1.1. MI.SourceExtended
      2.2.1.2. MI.LoadBalanceMetadata
    2.2.2. MI.Auth
      2.2.2.1. MI.HeaderAuth
      2.2.2.2. MI.AWSv4Auth
  2.3. Edge Control Metadata
    2.3.1. MI.CrossOriginPolicy
      2.3.1.1. MI.AccessControlAllowOrigin
    2.3.2. MI.AllowCompress
    2.3.3. MI.TrafficType
    2.3.4. MI.OcnSelection
      2.3.4.1. MI.OcnDelivery
  2.4. Processing Stage Metadata
    2.4.1. MI.ProcessingStages
    2.4.2. MI.StageRules
    2.4.3. MI.ExpressionMatch
    2.4.4. MI.StageMetadata
    2.4.5. MI.RequestTransform
    2.4.6. MI.ResponseTransform
    2.4.7. MI.SyntheticResponse

2.4.8. MI.HeaderTransform
2.4.9. MI.HTTPHeader
2.5. General Metadata
2.5.1. MI.ServiceIDs
2.5.2. MI.PrivateFeatureList
2.5.2.1. MI.PrivateFeature
2.5.3. MI.RequestRouting
3. Metadata Expression Language
3.1. Expression Variables
3.2. Expression Operators and keywords
3.3. Expression Built-in Functions
3.3.1. Basic Functions: Type Conversions
3.3.2. Basic Functions: String Conversions
3.3.3. Convenience Functions
3.4. Error Handling
3.4.1. Compile Time Errors
3.4.2. Runtime Errors
3.5. Expression Examples
3.5.1. ComputedCacheKey
3.5.2. ExpressionMatch
3.5.3. ResponseTransform
3.5.4. MI.ServiceIDs
4. CDNI Capabilities Extensions
4.1. FCI Metadata Object
4.2. FCI Model Extensions
4.2.1. FCI.AuthTypes
4.2.2. FCI.ProcessingStages
4.2.3. FCI.SourceMetadataExtended
4.2.4. FCI.RequestRouting
4.2.5. FCI.PrivateFeatures
4.2.5.1. FCI.PrivateFeature
4.2.6. FCI.OcnSelection
5. IANA Considerations
5.1. CDNI Payload Types
6. Security Considerations
7. Conclusion
8. References
8.1. Normative References
8.2. Informative References
Authors' Addresses

1. Introduction and Scope

The Content Delivery Network Interconnection (CDNI) Metadata interface enables interconnected Content Delivery Networks (CDNs) to exchange content distribution metadata in order to enable content acquisition and delivery. To facilitate a wider set of use cases encountered in the commercial CDN and Open Caching ecosystems, this

document describes extensions to the CDNI Metadata object model and its associated Capabilities model.

The objectives of this document are:

1. Identify the requirements for extending [RFC8006] and [RFC8008] and specify a set of extensions that realize these requirements.

2. Maintain backward compatibility with [RFC8006] and [RFC8008] by not altering the definitions or semantics of the original object model. All extensions are defined as new GenericMetadata Objects.

3. Define the metadata object model independently of the APIs used to publish and retrieve metadata.

Scope this document ADDRESSES:

1. Define and register CDNI GenericMetadata objects, as defined in section 4 of [RFC8006].

2. Define and register CDNI Payload Types, as defined in section 7.1 of [RFC8006].

3. Define Capabilities Objects that facilitate advertisement of a dCDN's support of these new metadata features, extending definitions in section 5 of [RFC8008].

4. Specification of a Metadata Expression Language Section 3 used within the metadata object model extensions.

5. Provide JSON examples illustrating real-world CDN and Open Caching use cases.

Scope this document DOES NOT ADDRESS:

1. Metadata object model definitions already specified in [RFC8006].

2. Interface API definitions for publishing and retrieving configuration metadata. The Metadata Interface (MI) as defined in [RFC8006] can be used to retrieve metadata. To enable more sophisticated metadata configuration publishing workflows, the Streaming Video Alliance (SVA) Open Caching API [OC-CI], as documented in the SVA Configuration Interface Part 3 (Simple API) and Part 4 (Advanced API) specifications can be used.

1.1. Terminology

For consistency with other CDNI documents this document follows the CDNI convention of uCDN (upstream CDN) and dCDN (downstream CDN). It should be noted, however, that uCDN and dCDN are roles that can be played by a variety of entities in the distribution ecosystem. A Content Provider, for example, can play the roles of a uCDN, while a commercial CDN or Open Caching system can play either the roles of a uCDN or dCDN. Additionally, this document reuses the terminology defined in [RFC6707],[RFC7336], [RFC8006], [RFC8007] and [RFC8804].

The following terms are used throughout this document:

*API - Application Programming Interface

*AWS - Amazon Web Services

*CDN - Content Delivery Network

*CDNi - CDN Interconnect

*CORS - Cross-Origin Resource Sharing

*CP - Content Provider

*dCDN - Downstream CDN

*DNS - Domain Name System

*FCI - Footprint and Capabilities Advertising Interface

*HREF - Hypertext Reference (link)

*HTTP - Hypertext Transfer Protocol

*IETF - Internet Engineering Task Force

*ISP - Internet Service Provider

*JSON - JavaScript Object Notation

*MEL - Metadata Expression Language

*Object - A collection of properties.

*OC - Open Caching

*OCN - Open Caching Node

*PatternMatch - An object which matches a string pattern

*UA - User Agent

*uCDN - Upstream CDN

*URI - Uniform Resource Identifier

*URN - Uniform Resource Name

*VOD - Video-on-Demand

*W3C - World Wide Web Consortium

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. CDNI Metadata Model Extensions

This section details extensions to the CDNI Metadata model as defined in Section 4 of [RFC8006], expressed as a set of new GenericMetadata objects. To preserve backward compatibility with [RFC8006], no changes are proposed to the original set of GenericMetadata.

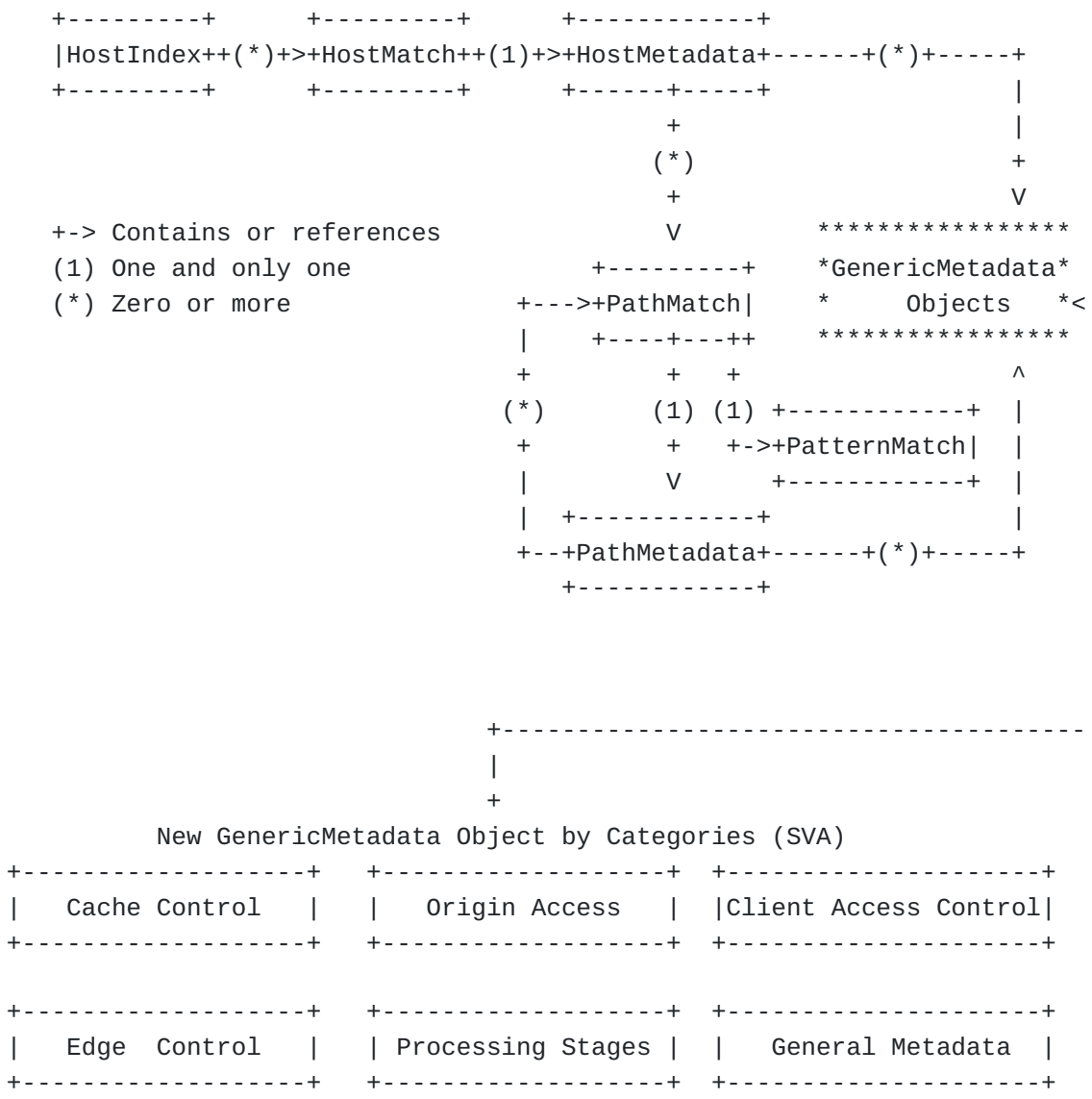

Figure 1: CDNI Metadata Model with Extensions

The remainder of this section presents the extended set of GenericMetadata objects organized by the categories in the above diagram.

Note: In the following sections, the term "mandatory-to-specify" is used to convey which properties MUST be included when serializing a given capability object. When mandatory-to-specify is defined as "Yes" for an individual property, it means that if the object containing that property is included in a message, then the mandatory-to-specify property MUST also be included.

2.1. Cache Control Metadata

In addition to the cache control policies currently specified by CDNI metadata, content providers often need more fine-grained

control over CDN caching, including scenarios where it is desirable to override or adjust cache-control headers from the origin.

The following additional capabilities are needed for general CDN and open caching use cases:

1. Positive Cache Control - Allows the uCDN to specify internal caching policies for the dCDN and external caching policies advertised to clients of the dCDN, overriding any cache control policy set in the response from the uCDN.

2. Negative Cache Control - Allows the specification of caching policies based on error response codes received from the origin, allowing for fine-grained control of the downstream 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 service or uCDN from being flooded with requests.

3. Cache Bypass Control - Allows content providers to bypass CDN caching when needed (typically for testing or performance benchmarking purposes).

4. 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 occur asynchronously.

5. Dynamically Constructed Cache Keys - While the properties provided by the standard CDNI metadata Cache object provide some simple control over the construction of the cache key, 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 origin may specify a cache key as a value returned in a specific HTTP response header. A rich expression language is provided to allow for such advanced cache key construction.

2.1.1. MI.CachePolicy

CachePolicy is a new GenericMetadata object that allows for the uCDN to specify internal caching policies for the dCDN, as well as external caching policies advertised to clients of the dCDN

(overriding any cache control policy set in the response from the uCDN).

Property: internal

-Description: Specifies the internal cache control policy to be used by the dCDN.

-Type: Number in seconds encoded as string (e.g. 5 is a five second cache) and/or a list of Enumeration [as-is|no-cache|no-store]

-Mandatory-to-Specify: No. The default is to use the cache control policy specified in the response from the uCDN.

Property: external

-Description: Specifies the external cache control policy to be used by clients of this dCDN.

-Type: Number in seconds encoded as string (e.g. 5 is a five second cache) and/or a list of Enumeration [as-is|no-cache|no-store]

-Mandatory-to-Specify: No. The default is to use the cache control policy specified in the response from the uCDN.

Property: force

-Description: If set to True, the metadata interface cache policy defined in the MI.CachePolicy will override any cache control policy set in the response from the uCDN. If set to False, the MI.CachePolicy 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 only if no policy is provided in the response from the uCDN.

Example 1: An MI.CachePolicy that sets the internal cache control policy to five seconds. The external cache policy is set to 'no-cache':

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

Example 2: An MI.CachePolicy that 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:

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

Example 3: An MI.CachePolicy in the context of the processing stages model that sets a caching policy only if the HTTP status code received from the origin is a 200. In this example, the internal cache control policy is set to five seconds. The external cache policy is set to 'no-cache'. Force is set to 'False', indicating that the MI.CachePolicy only applies if there is no cache policy in the response from the uCDN.

```
{
  "generic-metadata-type": "MI.ProcessingStages",
  "generic-metadata-value": {
    "origin-response": [
      {
        "match": {
          "expression": "resp.status == 200"
      },
      "stage-metadata": {
          "generic-metadata": [
            {
              "generic-metadata-type": "MI.CachePolicy",
              "generic-metadata-value": {
                "internal": "5",
                "external": "no-cache",
                "force": "false"
              }
            }
          ]
        }
      }
    ]
  }
}
```

2.1.2. MI.NegativeCachePolicy

NegativeCachePolicy is a new GenericMetadata object that allows for the specification of caching policies based on error response codes received from the origin.

Property: error-codes

-Description: Array of HTTP response error status codes (See Sections 6.5 and 6.6 of [RFC7231] , that if returned from the uCDN, will be cached using the cache policy defined by the cache-policy property.

-Type: Array of HTTP response error status codes

-Values: ["404", "503", "504"]

-Mandatory-to-Specify: No. The default is to revert to [RFC8006] behavior.

Property: cache-policy

-Description: MI.CachePolicy to apply to the HTTP response error status codes returned by the uCDN.

-Mandatory-to-Specify: Yes

Example: A MI.NegativeCachePolicy that applies to HTTP error codes: "404", "503", "504" and sets the internal cache control policy to five seconds and external to 'no-cache'.

```
{
  "generic-metadata-type": "MI.NegativeCachePolicy",
  "generic-metadata-value": {
    "error-codes": [ "404", "503", "504" ],
    "cache-policy": {
      "internal": "5",
      "external": "no-cache",
      "force": "true"
    }
  }
}
```

2.1.3. MI.StaleContentCachePolicy

MI.StaleContentCachePolicy is a new GenericMetadata object that allows the uCDN to specify the policy to use by a dCDN when responding with 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 occur asynchronously.

Property: stale-while-revalidating

-Description: Instructs the dCDN to serve a stale version of a resource while refreshing the resource with the uCDN. When set to "True", the dCDN will return a previously cached version of a resource while the resource is refreshed with the uCDN in the background.

-Type: Boolean

-Mandatory-to-Specify: No. The default is False, which waits for the uCDN to refresh a resource before responding to the client.

Property: stale-if-error

-Description: Instructs the dCDN to serve a stale version of a resource if an error was received when trying to refresh the resource with the uCDN. When set, the dCDN will return a previously cached version of a resource instead of caching the error response. Per Section 4 of [RFC5861], an error is any situation that would result in a 500, 502, 503, or 504 HTTP response status code being returned

-Type: Array of HTTP response error status codes. Example: ["503", "504"]

-Mandatory-to-Specify: No. The default is to cache the error response received from the uCDN.

Property: failed-refresh-ttl

-Description: Instructs the dCDN to serve a stale version of a resource for the number of seconds specified in failed-refresh-ttl before trying to revalidate the resource with the uCDN. Use of failed-refresh-ttl allows the load to be reduced on the uCDN during times of system stress.

-Type: Integer

-Mandatory-to-Specify: No

Example 1: A MI.StaleContentCachePolicy where stale-while-revalidating is true, instructing the dCDN to respond with a stale cached version of the resource while it refreshes the resource with the uCDN in the background:

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

Example 2: A 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 refresh the resource with the uCDN.

failed-refresh-ttl instructs the dCDN to use a five second cache TTL on the resource that receives an error when refreshing from the uCDN. That is, after five seconds, the dCDN will attempt to refresh the resource with the uCDN.

```
{
  "generic-metadata-type": "MI.StaleContentCachePolicy",
  "generic-metadata-value": {
    "stale-if-error": [ "503", "504" ],
    "failed-refresh-ttl": "5"
  }
}
```

Example 3: A MI.StaleContentCachePolicy where stale-while-revalidating is true, instructing the dCDN to respond with a stale

cached version of the resource while it refreshes 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, 503, or 504 when trying to refresh the resource with the uCDN.

failed-refresh-ttl instructs the dCDN to use a five second cache TTL on the resource that receives an error when refreshing from the uCDN. That is, after five seconds, the dCDN will attempt to refresh the resource with the uCDN.

```
{
  "generic-metadata-type": "MI.StaleContentCachePolicy",
  "generic-metadata-value": {
    "stale-while-revalidating": "true",
      "stale-if-error": [ "404", "503", "504" ],
      "failed-refresh-ttl": "5"
  }
}
```

2.1.4. MI.CacheBypassPolicy

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 cache when testing the uCDN fill path. Note: 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 CacheBypassPolicy) is not 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 to allow a client request to be marked to bypass cache.

-Type: Boolean

-Mandatory-to-Specify: No. The default is False.

Example 1: A MI.CacheBypassPolicy with the client HTTP header of: CDN-BYPASS: "True":

```
{
  "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"
              }
            }
          ]
        }
      }
    ]
  }
}
```

Example 2: A MI.CacheBypassPolicy that applies to all requests where the host header is bypass.example.com:

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

2.1.5. MI.ComputedCacheKey

While the properties provided by the standard CDNi metadata Cache object (See Section 4.2.6 of [RFC8006]) provide some simple control over the construction of the cache key, 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 origin may specify a cache key as a value returned in a specific HTTP response header.

ComputedCacheKey is a new GenericMetadata object that allows for the specification of a cache key using the metadata expression language. Typical use cases would involve the construction of a cache key from one or more elements of the HTTP request. In cases where both the ComputedCacheKey and the Cache object are applied, the ComputedCacheKey will take precedence.

Property: expression

-Description: The expression that specifies how the cache key shall be constructed.

-Type: String. An expression using [CDNI-MEL] to dynamically construct the cache key from elements of the HTTP request and/or response.

-Mandatory-to-Specify: Yes

Example, using a custom request header as the cache key instead of the URI path:

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

2.2. Origin Access Metadata

The CDNI metadata definitions for sources (also known as origins in the CDN industry), are extended to provide the following capabilities required:

1. Designation as to whether a source requires HTTPS access.

2. Specification of the source's TCP port number.

3. Web root path specification for the source.

4. Indication as to whether redirects should be followed.

5. Support for additional forms of origin authentication.

6. Multi-origin failover - The ability to specify a list of origins that can act as fallbacks to the primary origin. Failure rules can specify types of errors and timeout values that trigger failover.

7. Multi-origin load balancing - The ability to specify a list of origins that can be selected by one of several balancing rules (round robin, content hash, IP hash).

8. Specification of SNI configurations required for origin access.

9. Specification of connection control parameters for origin access.

2.2.1. MI.SourceMetadataExtended

SourceMetadataExtended is an alternative to the CDNI standard SourceMetadata object, which adds a property to specify load balancing across multiple sources, as well as a SourceExtended sub-object with additional attributes to the CDNI standard Source object. While both SourceMetadataExtended and SourceMetadata can be provided for backward compatibility, a dCDN that advertises capability for SourceMetadataExtended will ignore SourceMetadata if both are provided for a given host or path match.

Property: sources

-Description: Sources from which the dCDN can acquire content, listed in order of preference.

-Type: Array of SourceExtended objects

-Mandatory-to-Specify: No. Default is to use static configuration, out-of-band from the CDNI metadata interface.

Property: load-balance

-Description: Specifies load balancing rules for the set of sources.

-Type: LoadBalanceMetadata object

-Mandatory-to-Specify: No

Example of a SourceMetadataExtended object with the associated LoadBalanceMetadata configuration object:

```
{
  "generic-metadata-type": "MI.SourceMetadataExtended",
  "generic-metadata-value": {
    "sources": [
      {
        "endpoints": [
          "a.service123.ucdn.example",
          "b.service123.ucdn.example"
        ],
        "protocol": "http/1.1"
      },
      {
        "endpoints": [
          "origin.service123.example"
        ],
        "protocol": "http/1.1"
      }
    ],
    "load-balance": {
      "balance-algorithm": "content-hash",
      "balance-path-pattern": "^/prod/(.*)/.*\\.ts$"
    }
  }
}
```

2.2.1.1. MI.SourceExtended

SourceExtended is an alternative to the CDNI standard Source object with additional metadata. It contains all the attributes of the [RFC8006] Source object (acquisition-auth, endpoints, and protocol), with additions specified below.

Property: acquisition-auth

-Description: Authentication method to use when requesting content from this source. Same as [RFC8006].

-Type: Auth (see [RFC8006] Section 4.2.7 and the new MI.Auth types in this specification)

-Mandatory-to-Specify: No. Default is no authentication required.

Property: endpoints

-Description: Origins from which the dCDN can acquire content. If multiple endpoints are specified, they are all equal, i.e., the list is not ordered by preference. Same as [RFC8006].

-Type: Array of Endpoint objects (see [RFC8006] Section 4.3.3)

-Mandatory-to-Specify: Yes..

Property: protocol

-Description: Network retrieval protocol to use when requesting content from this source. Same as [RFC8006].

-Type: Protocol (see [RFC8006] Section 4.3.2)

-Mandatory-to-Specify: Yes..

Property: origin-host

-Description: HTTP host header to pass to the endpoints when retrieving content from a uCDN. The host MUST conform to the Domain Name System (DNS) syntax defined in [RFC1034] and [RFC1123]

-Type: String

-Mandatory-to-Specify: No. The default is to use the host name passed by the dCDN.

Property: webroot

-Description: The path element that is prepended to a resource's URI before retrieving content from a uCDN.

-Type: String

-Mandatory-to-Specify: No. The default is to use the original URI.

Property: follow-redirects

-Description: If the follow-redirects property is set to "True", HTTP redirect responses returned from a uCDN will be followed when retrieving content. Otherwise, the HTTP redirect response is returned to the client.

-Type: Boolean

-Mandatory-to-Specify: No. The default is "True" (i.e., follow redirect responses from the uCDN).

Property: timeout-ms

-Description: A timeout (in milliseconds) to apply when connecting to a uCDN. If the connection is not established within timeout-ms, this source is abandoned and the next

source in the MI.SourceMetadataExtended sources array is tried. Once a connection is established, timeout-ms is used on subsequent reads of data from the uCDN.

-Type: Integer

-Mandatory-to-Specify: No. The default is to revert to [RFC8006] behavior.

Property: failover-errors

-Description: Array of HTTP response error status codes (Section 6 of [RFC7231]), that if returned from the uCDN, will trigger a failover to the next source in the MI.SourceMetadataExtended sources array. If the uCDN returns an HTTP error code that is not in the failover-errors array, that error code is returned to the client of the dCDN.

-Type: Array of HTTP response error status codes.

-Mandatory-to-Specify: No. The default is to revert to [RFC8006] behavior.

Example of a SourceExtended object that describes a pair of endpoints (servers) that the dCDN can use to acquire content for the applicable host and/or URI path:

```
{
  "generic-metadata-type": "MI.SourceMetadataExtended",
  "generic-metadata-value": {
    "sources": [
      {
        "endpoints": [
          "a.service123.ucdn.example",
          "b.service123.ucdn.example:8443"
        ],
        "protocol": "https/1.1",
        "origin-host": "internal.example.com",
        "webroot": "/prod",
        "follow-redirects": false,
        "timeout-ms": 4000,
        "failover-errors": [ "502", "503", "504" ]
      },
      {
        "endpoints": [ "origin.service123.example" ],
        "protocol": "http/1.1",
        "webroot": "/prod",
        "follow-redirects": true,
        "timeout-ms": 8000
      }
    ]
  }
}
```

2.2.1.2. MI.LoadBalanceMetadata

The LoadBalanceMetadata object defines how content acquisition requests are distributed over the SourceExtended objects listed in the SourceMetadataExtended object.

Property: balance-algorithm

-Description: Specifies the algorithm to be used when distributing content acquisition requests over the sources in a SourceMetadataExtended object. The available algorithms are random, content-hash, and ip-hash.

o random: Requests are distributed over the sources in proportion to their associated weights.

o content-hash: Requests are distributed over the sources in a consistent fashion, based on the balance-path-pattern property.

o ip-hash: Requests are distributed over the sources in a consistent fashion based on the IP address of the client requestor.

1. Type: Enumeration [random|content-hash|ip-hash] encoded as a lowercase string.

2. Mandatory-to-Specify: No. The default is to use sources in preference order as defined in the SourceMetadataExtended object.

Property: balance-weights

-Description: This property specifies the relative frequency that a source is used when distributing requests. For example, if there are three SourceExtended objects in a SourceMetadataExtended object with balance-weights [1, 2, 1], source 1 will receive 1/4 of the requests; source 2 will receive 2/4 of the requests; source 3 will receive 1/4 of the requests.

-Type: Array of integers

-Mandatory-to-Specify: No. The default is to use sources in preference order as defined in the SourceMetadataExtended object.

Property: balance-path-pattern

-Description: This property specifies a regular expression pattern to apply to the URI when calculating the content hash used by the balance-algorithm. For example, "balance-path-pattern": "^/prod/(.*)/.*\.ts$" would distribute requests based on the URI but excluding the /prod prefix and the .ts segment file.

-Type: String (regular expression)

-Mandatory-to-Specify: No. The default is to use the original URI.

Example 1: The LoadBalanceMetadata object distributes content acquisition requests over sources using a content-hash algorithm:

```
{
  "generic-metadata-type": "MI.LoadBalanceMetadata",
  "generic-metadata-value": {
    "balance-algorithm": "content-hash",
    "balance-path-pattern": "^/prod/(.*)/.*\\.ts$"
  }
}
```

Example 2: The LoadBalanceMetadata object distributes content acquisition requests over sources using the random algorithm:

```
{
  "generic-metadata-type": "MI.LoadBalanceMetadata",
  "generic-metadata-value": {
    "balance-algorithm": "random",
    "balance-weights": [ 1, 2, 1 ]
  }
}
```

2.2.2. MI.Auth

To meet the typical industry requirements for authenticating CDNs to external origins, two new authentication types are defined.

auth-type: MI.HeaderAuth

- Description: Header based authentication is used to pass an HTTP header (secret-name) and value (secret-value) to a uCDN when requesting content. The header name and value are agreed upon between parties out of band.

- Note: We may want to add a way to encrypt or separately communicate the secret; this could be a general capability for CDNI.

- auth-value: MI.HeaderAuth object specifying the header name and value (secret name, secret key) required for authenticated access to an origin. For more information, refer to the MI.HeaderAuth section below.

auth-type: MI.AWSv4Auth

- Description: Allows for the specification of a set of headers to be added to requests that are forwarded to an origin to enable Amazon Web Services (AWS) authentication, as documented by AWS (See Specifications & Standards References).

- auth-value: MI.AWSv4Auth object specifying the access parameters. For more information, refer to the MI.AWSv4Auth section below.

2.2.2.1. MI.HeaderAuth

The HeaderAuth metadata object is used in the auth-value property of the

Auth object, as defined in [RFC8006] section 4.2.7, and may be applied to any

source by including or referencing it under its authentication property. This method of authentication provides a simple capability for a mutually agreed upon header to be added by the CDN to all requests sent to a specific origin. Note that if a dynamically generated header value is required, the RequestTransform capabilities within StageProcessing can be used.

Property: header-name

-Description: Name of the authentication header.

-Type: String

-Mandatory-to-Specify: Yes

Property: header-value

-Description: Value of the authentication header (typically a pre-shared key). Note that this value SHOULD NOT be disclosed; it SHOULD be protected by some mechanism such as a secret-sharing API, which is outside the scope of this specification.

-Type: String

-Mandatory-to-Specify: Yes

Example Auth object for header authentication:

```
{
  "generic-metadata-type": "MI.SourceMetadataExtended",
  "generic-metadata-value": {
    "sources": [
      {
        "endpoints": [ "origin.example.com" ],
        "protocol": "http/1.1",
        "acquisition-auth": {
          "generic-metadata-type": "MI.Auth",
          "generic-metadata-value": {
            "auth-type": "MI.HeaderAuth",
            "auth-value": {
              "header-name": "X-Origin-Auth",
              "header-value": "SECRETKEYJKSDHFSIFUI4UFH78HW4NF7"
            }
          }
        }
      }
    ]
  }
}
```

2.2.2.2. MI.AWSv4Auth

The AWSv4Auth metadata object is used in the auth-value property of the Auth object as defined in [RFC8006] section 4.2.7, and may be applied to any source by including or referencing it under its authentication property.

AWSv4 authentication causes upstream requests to have a signature applied, following the method described in [AWSv4Method]. A hash-based signature is calculated over the request URI and specified headers, and provided in an Authorization: header on the upstream request. The signature is tied to a pre-shared secret key specific to an AWS service, region, and key ID.

1. We may want to add a way to encrypt or separately communicate the secret; this could be a general capability for CDNI.

2. We may want to add optional properties that allow overriding the default headers to sign.

3. We may want to add optional properties that allow the signature to be sent in a way other than with the Authorization: header (e.g., query strings are also supported).

Property: access-key-id

-Description: The preconfigured ID of the pre-shared authorization secret.

-Type: String

-Mandatory-to-Specify: Yes

Property: secret-access-key

-Description: The pre-shared authorization secret, which is the basis of building the signature. This is a secret key that SHOULD NOT be disclosed; it SHOULD be protected by some mechanism such as a secret-sharing API, which is outside the scope of this specification.

-Type: String

-Mandatory-to-Specify: Yes

Property: aws-region

-Description: The AWS region name that is hosting the service and shares the key ID and corresponding pre-shared secret.

-Type: String

-Mandatory-to-Specify: Yes

Property: aws-service

-Description: The AWS service name that is serving the upstream requests.

-Type: String

-Mandatory-to-Specify: No. It defaults to "s3" if not specified.

Property: host-name

-Description: The host name to use as part of the signature calculation.

-Type: String

-Mandatory-to-Specify: No. It defaults to using the value of the Host: header of the upstream request. This property is available in case the application needs to override that behavior.

Example Auth object for AWSv4 authentication:

```
{
  "generic-metadata-type": "MI.SourceMetadataExtended",
  "generic-metadata-value": {
    "sources": [
      {
        "endpoints": [ "origin.example.com" ],
        "protocol": "http/1.1",
        "acquisition-auth": {
          "generic-metadata-type": "MI.Auth",
          "generic-metadata-value": {
            "auth-type": "MI.AWSv4Auth",
            "auth-value": {
              "access-key-id": "MYACCESSKEYID",
              "secret-access-key": "SECRETKEYJKSDHFSIUHKWRGHHF",
              "aws-region": "us-west-1"
            }
          }
        }
      }
    ]
  }
}
```

2.3. Edge Control Metadata

CDNs typically require a set of configuration metadata to inform processing of responses downstream (at the edge and in the user agent). This section specifies GenericMetadata objects to meet those requirements.

2.3.1. MI.CrossOriginPolicy

Delegation of traffic between a CDN over an open caching node based on HTTP redirection does change the domain name in the client requests. This represents a cross-origin request that must be managed appropriately using Cross-Origin Resource Sharing (CORS) headers in the responses.

The dynamic generation of CORS headers is typical in modern HTTP request processing and avoids CORS validation forwarded to the CDN origin servers, particularly with the preflight OPTIONS requests. The CDNI metadata model requires extensions to specify how a CDN or open caching node should generate and evaluate these headers.

Required capabilities:

1. Set a default value for CORS response headers independent of the origin request header value.

2. Match the origin request header with a list of valid values, including PatternMatch, to return or not return the CORS response headers.

3. Set a list of custom headers that can be exposed to the client (expose headers).

4. Support for preflight requests using the OPTIONS method, including custom header validation, expose headers, and methods.

5. Support for credentials validation within CORS.

Simple CORS requests are those where both HTTP method and headers in the request are included in the safe list defined by the World Wide Web Consortium [W3C]. The user agent (UA) request can include an origin header set to the URL domain of the webpage that the player runs. Depending on the metadata configuration, the logic to apply by the open caching node (OCN) is:

1. Validation of the origin header - Metadata can include a list of valid domains to validate the request origin header. If it does not match, the CORS header must not be included in the response.

2. WIldcard usage - Depending on the configuration, the resultant CORS header to include in the response will be the same as the request origin header, or a wildcard.

3. If no validation of request is included in the origin header, set a default value for CORS response headers independent of the origin request header value.

When a UA makes a request that includes a method or headers that are not included in the safe-list, the client will make a CORS preflight request using the OPTIONS method to the resource including the origin header. If CORS is enabled and the requests passes the origin validation, the OCN SHOULD respond with the set of headers that indicate what is permitted for that resource, including one or more of the following:

1. Allowed methods

2. Allowed credentials

3. Allowed request headers

4. Max age that the OPTIONS request is valid

5. Headers that can be exposed to the client

CrossoriginPolicy is a GenericMetadata object that allows for the specification of dynamically generated CORS headers.

Property: allow-origin

-Description: Validation of simple CORS requests.

-Type: Object MI.AccessControlAllowOrigin

-Values: One element for each of the following properties.

-Mandatory-to-Specify: Yes

Property: expose-headers

-Description: A list of values the OCN will include in the Access-Control-Expose-Headers response header to a preflight request.

-Type: Array of strings

-Mandatory-to-Specify: No

Property: allow-methods

-Description: A list of values the OCN will include in the Access-Control-Allow-Methods response header to a preflight request.

-Type: Array of strings

-Mandatory-to-Specify: No

Property: allow-headers

-Description: A list of values the OCN will include in the Access-Control-Allow-Headers response header to a preflight request.

-Type: Array of strings

-Mandatory-to-Specify: No

Property: allow-credentials

-Description: The value the OCN will include in the Access-Control-Allow-Credentials response header to a preflight request.

-Type: Boolean

-Mandatory-to-Specify: No

Property: max-age

-Description: The value the OCN will include in the Access-Control-Max-Age response header to a preflight request.

-Type: Integer

-Mandatory-to-Specify: No

2.3.1.1. MI.AccessControlAllowOrigin

The MI.AccessControlAllowOrigin object has the following properties:

Property: allow-list

-Description: List of valid URLs that will be used to match the request origin header. The Origin header is a HTTP extension. Its value is a version of the Referer header in some specific requests, and used for Cross Origin requests. . Permitted values are schema://hostname[:port]

-Type: Array of PatternMatch objects

-Mandatory-to-Specify: Yes

Property: wildcard-return

-Description: If "True", the OCN will include a wildcard (*) in the Access-Control-Allow-Origin response header. If "False", the OCN will reflect the request origin header in the Access-Control-Allow-Origin response header.

-Type: Boolean

-Mandatory-to-Specify: Yes

The examples below demonstrate how to configure response headers dynamically for CORS validation.

Example 1: A simple CORS validation configuration:

```
{
  "generic-metadata-type": "MI.CrossoriginPolicy",
  "generic-metadata-value": {
    "allow-origin": {
      "allow-list": [
        {
          "pattern": "*"
        }
      ],
      "wildcard-return": true
    }
  }
}
```

Example 2: Validation of a preflight request when some of the headers included in the subsequent object request are not included in the CORS specification safelist:

```
{
  "generic-metadata-type": "MI.CrossoriginPolicy",
  "generic-metadata-value": {
    "allow-origin": {
      "allow-list": [
        {
          "pattern": "*://sourcepage.example.com"
        },
        "wildcard-return": false
      },
      "allow-methods": [ "GET", "POST" ],
      "allow-credentials": true,
      "allow-headers": [ "X-PINGOTHER", "Content-Type" ],
      "expose-headers": [ "X-User", "Authorization" ],
      "max-age": 3600
    }
  }
}
```

2.3.2. MI.AllowCompress

Downstream CDNs often have the ability to compress HTTP response bodies in cases where the client has declared that it can accept compressed responses (via an Accept-Encoding header), but the source/origin has returned an uncompressed response.

The specific compression algorithm used by the dCDN is negotiated by the client's Accept-Encoding header according to [RFC7694] (including q= preferences) and the compression capabilities available on the dCDN.

In addition, HeaderTransform allows the uCDN to normalize, or modify, the Accept-Encoding header to allow for fine-grain control over the selection of the compression algorithm (e.g., gzip, compress, deflate, br, etc.).

AllowCompress is a new GenericMetadata object that allows the dCDN to compress content before sending to the client.

Property: allow-compress

-Description: If set to "True", then the dCDN will try to compress the response to the client based on the Accept-Encoding request header.

-Type: Boolean

-Values: True or False

-Mandatory-to-Specify: No. The default is "False".

Example 1: An MI.AllowCompress that allows manifests (*.m3u8) to be compressed by the dCDN:

```
{
  "match": {
    "expression": "req.h.uri *= '*.m3u8'"
  },
  "stage-metadata": {
    "generic-metadata": [
      {
        "generic-metadata-type": "MI.AllowCompress",
        "generic-metadata-value": {
          "allow-compress": "true"
        }
      }
    ]
  }
}
```

Example 2: An MI.AllowCompress that allows manifests (*.m3u8) to be compressed by the dCDN but normalizing the client's Accept-Encoding header:

```
{
  "match": {
    "expression": "req.h.accept-encoding *= '*gzip*'"
  },
  "stage-metadata": {
    "generic-metadata": [
      {
        "generic-metadata-type": "MI.AllowCompress",
        "generic-metadata-value": {
          "allow-compress": "true"
        }
      }
    ]
  }
}
```

2.3.3. MI.TrafficType

Content delivery networks often apply different infrastructure, network routes, and internal metadata for different types of traffic. Delivery of large static objects (such as software downloads), may, for example, use different network routes than video stream delivery. In an HTTP adaptive bitrate video service, every video title corresponds to a set of video files and descriptors according to different video protocols, and this is

independent of the type of service (Video-on-Demand, Live, Catch-up, etc.).

The way the video service is consumed by the user agents can vary. For instance, a segment that belongs to a Video-on-Demand (VoD) title can be requested for every moment the content is available for the user agents to consume, while a segment of live content will be only requested as long as the time-shift duration is configured for that service. Knowing those differences, a CDN or OCN provider can implement specific strategies that will maximize performance and thereby provide more available capacity to the upstream provider. It should be noted that the dCDNs handling of the traffic types is implementation-specific and not prescribed here.

TrafficType metadata defines a set of descriptors that characterize either the type or usage of the traffic, enabling CDNs and OCNs to apply any internal configuration rules without exposing an unnecessary number of internal details. Note that the interpretation of these traffic types and application of rules such as rate limiting or delivery pacing are implementation specific.

Property: traffic-type

-Description: A literal that defines the traffic type. uCDN will use the literal that is most representative of the traffic being delegated.

-Type: Enumeration [vod, live, object-download] encoded as lowercase string

-Mandatory-to-Specify: Yes

Property: hints

-Description: Other traffic characteristics that the uCDN can indicate to the dCDN as suggestions for service optimization. Accepts free-form unconstrained values.

-Type: Array of strings

-Mandatory-to-Specify: No

A TrafficType definition example for HostMetadata:

```
{
  "generic-metadata-type": "MI.TrafficType",
  "generic-metadata-value": {
    "traffic-type": "vod",
    "hints": [ "low-latency", "catch-up" ]
  }
}
```

2.3.4. MI.OcnSelection

Configuration metadata is required to permit several levels of OCN selection policies. For example, in a mobile network, several physical locations are possible (i.e., candidates) for hosting the OCN that will take charge in the delegation for the uCDN. This is the case when the cache is virtualized and deployed dynamically. Depending on the OCN selection policy (which may be a cost driver), the dCDN may attempt to favor certain types of caches at the edge, for example. The default OCN selection policy might be "best-effort". Another one might be linked to the network characteristics like "Edge" or ("average latency< 10ms").

OcnSelection is a new GenericMetadata object that allows the uCDN to indicate to the dCDN a preference in terms of OCN selection.

Property: ocn-delivery

-Description: Instructs the dCDN to perform delegation operating a particular medium and/or a transport arrangement.

-Type: Object MI.OcnDelivery

-Mandatory-to-Specify: No. At least one of the two properties, ocn-type or ocn-delivery, must be present.

Property: ocn-type

-Description: Instructs the dCDN to perform delegation operating the type of open caching nodes.

-Type: A string corresponding to one of the open caching node types announced by the dCDN through the FCI interface.

-Mandatory-to-Specify: No. At least one of the two properties, ocn-type or ocn-delivery, must be present.

Property: ocn-selection

-Description: This property enforces the selection of OCNs, considering the ocn-type and/or the ocn-delivery properties. "False" means best-effort.

-Type: string. "attempt-or-failed" and "attempt-or-besteffort" mean that the delegation must be attempted considering the ocn-type and/or the ocn-delivery properties. If not possible, it is considered as an error and either fails (configuration failure) or the dCDN continues with a best-effort procedure. Last, "best effort" means the dCDN tries its best to fulfil the requested ocn-selection policy.

-Mandatory-to-Specify: No. Best-effort is the default OCN selection policy.

2.3.4.1. MI.OcnDelivery

An ocn-delivery object contains the following properties:

Property: ocn-medium

-Description: Instructs the dCDN to perform delegation operating a particular medium. The following values are specified: "SATELLITE".

-Type: String

-Mandatory-to-Specify: No. Either the ocn-medium property or the ocn-transport property must be present.

Property: ocn-transport

-Description: Instructs the dCDN to perform delegation operating a particular transport arrangement. The following values are specified: "MABR".

-Type: String

-Mandatory-to-Specify: No. At least one of the two properties (ocn-medium or ocn-transport) must be present.

2.4. Processing Stage Metadata

It is typical in CDN configurations to define matching rules and metadata that are to be applied at specific stages in the 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.

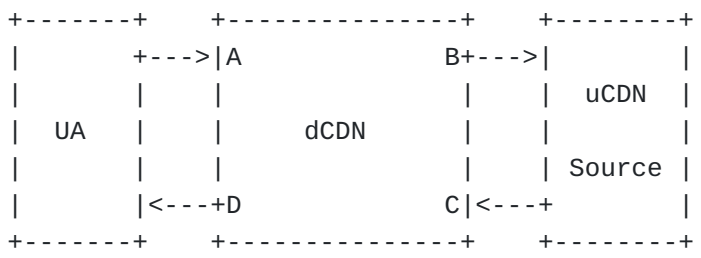

Figure 2: Processing stages

Processing stages:

1. clientRequest - Rules run on the client request prior to further processing.

2. originRequest - Rules run prior to making a request to the origin.

3. originResponse - Rules run after a response is received from the origin and before being placed in the cache.

4. clientResponse - Rules run prior to sending the 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.

Requirements:

1. 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 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 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.

2. 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.

3. 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.

4. 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. All of the following capabilities are required at each processing stage:

5. 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.

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

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

8. 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.

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

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

11. 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.

2.4.1. MI.ProcessingStages

A ProcessingStages object is a new GenericMetadata which describes the matching rules, metadata, and transformations to be applied at specific stages in the request processing pipeline. The processing rules and transformations are defined as a child data model referenced within a ProcessingStages object, as defined below.

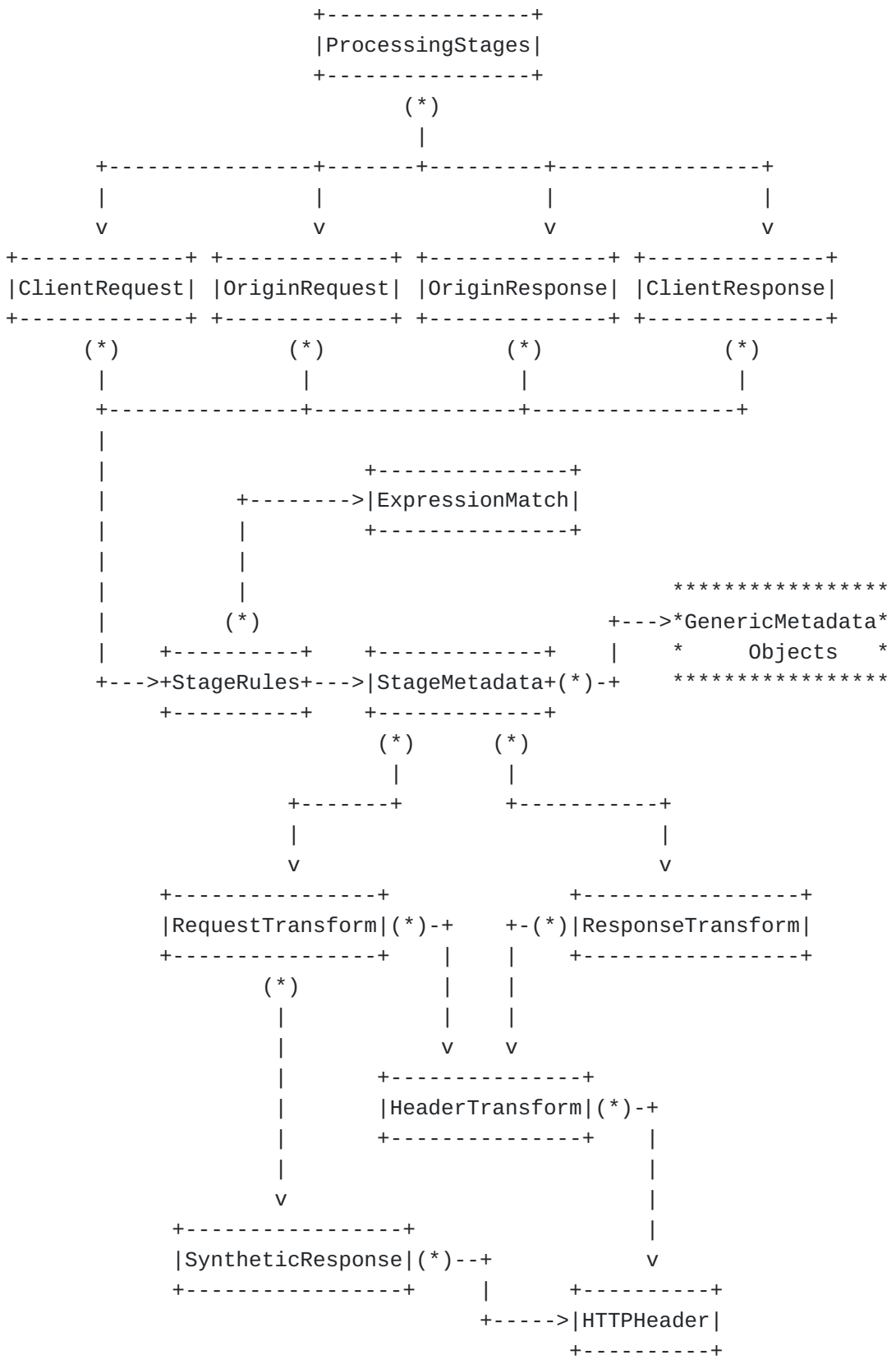

Figure 3: CDNi ProcessingStages metadata model with contained objects

Each of the four processing stages is represented by an array of StageRules objects, with each StageRules object defining criteria along with metadata that MUST be applied if the match applies to "True". It should be noted that the StageRules objects in the array are evaluated and processed in order. A possible future extension to this processing model could allow for an if-else structure, where processing for a stage is halted upon the first match of a StageRules expression.

Property: client-request

-Description: Allows for the specification of conditional metadata to be applied at the client request processing stages, as defined in the Rule Processing Stages section. The StageRules in the array are evaluated in order.

-Type: Array of StageRules objects

-Mandatory-to-Specify: No

Property: origin-request

-Description: Allows for the specification of conditional metadata to be applied at origin request processing stages, as defined in the Rule Processing Stages section. The StageRules in the array are evaluated in order.

-Type: Array of StageRules objects

-Mandatory-to-Specify: No

Property: origin-response

-Description: Allows for the specification of conditional metadata to be applied at origin response processing stages, as defined in the Rule Processing Stages section. The StageRules in the array are evaluated in order.

-Type: Array of StageRules objects

-Mandatory-to-Specify: No

Property: client-response

-Description: Allows for the specification of conditional metadata to be applied at client response processing stages, as defined in the Rule Processing Stages section. The StageRules in the array are evaluated in order.

-Type: Array of StageRules objects

-Mandatory-to-Specify: No

Example specifying all four processing stages. In this example, the client-request stage has two 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" : <Additional ExpressionMatch2 for conditional metadata
        "stage-metadata" : <StageMetadata2 for clientRequest stage>,
      }
    ],
    "origin-request" : [{
      "match" : <Optional ExpressionMatch for conditional metadata >
      "stage-metadata" : <StageMetadata for originRequest stage>,
    }],
    "origin-response" : [{
      "match" : <Optional ExpressionMatch for conditional metadata >
      "stage-metadata" : <StageMetadata for originResponse stage>,
    }],
    "client-response" : [{
      "match" : <Optional ExpressionMatch for conditional metadata >
      "stage-metadata" : <StageMetadata for clientResponse stage>,
    }]
  }
}
```

2.4.2. MI.StageRules

A 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.

Property: match

-Description: An ExpressionMatch object encapsulating a rich expression using the CDNI Metadata Expression Language [CDNI-MEL] 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 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 set of StageMetadata to be applied at the processing stage if the match expression evaluates to "True" or is not present.

-Type: Array of StageMetadata objects, applied in order.

-Mandatory-to-Specify: Yes

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).

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

2.4.3. MI.ExpressionMatch

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

Property: expression

-Description: A rich expression using CDNI-MEL to evaluate aspects of the HTTP request and/or response. See documentation on the Metadata Expression Language for details on the expression of matching variables and syntax.

-Type: String, using CDNI-MEL syntax. See the METADATA EXPRESSION LANGUAGE (CDNI-MEL) section.

-Mandatory-to-Specify: Yes

Example of ExpressionMatch on the referrer and user agent request headers:

```
{
  "expression" : "req.h.user-agent *= '*Safari*' and req.h.referrer == '
}
```

2.4.4. MI.StageMetadata

The 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: StageMetadata stages

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. Support for this capability is optional and can be advertised via feature-flags in the FCI interface.

-Type: Array of GenericMetadata, applied in order. Note that not all GenericMetadata object types may be applicable at all processing stages.

-Mandatory-to-Specify: No. The generic-metadata 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 via feature-flags in the FCI interface.

-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 via feature-flags in the FCI interface.

-Type: ResponseTransform object

-Mandatory-to-Specify: No

Example of a StageMetadata object:

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

2.4.5. MI.RequestTransform

The 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: headers

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

-Type: HeaderTransform object

-Mandatory-to-Specify: No

Property: uri

-Description: Replacement value for the HTTP request.

-Type: String. Either a literal (static string) or an expression using CDNI-MEL to dynamically construct a URI value from elements of the HTTP request and/or response.

-Mandatory-to-Specify: No

Property: uri-is-expression

-Description: Flag to signal whether the URI is a static string literal or a CDNI-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.

Example of a RequestTransform object illustrating a dynamically constructed URI rewrite:

```
{
  "request-transform" : {
    "headers" : { <Optional list of request headers to add/replace/delet
    "uri" :    "req.uri.path",
    "uri-is-expression" : true
 }
```

2.4.6. MI.ResponseTransform

The 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: headers

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

-Type: HeaderTransform object

-Mandatory-to-Specify: No

Property: response-status

-Description: Replacement value for the HTTP response status code.

-Type: Integer. Either a static integer or an expression using CDNI-MEL 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: Flag to signal whether the response-status is a static integer or a CDNI-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: 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

Example of a ResponseTransform object, illustrating a dynamically constructed header value that uses the expression language to concatenate the user agent and host header, and forces a 403 HTTP response status code:

```
{
  "response-transform": {
    "headers": {
      "add": [
        {
          "name": "X-custom-response-header",
          "value": "req.h.user-agent . &#8216;-&#8216; . req.h.host",
          "value-is-expressions": true
        }
      ]
    },
    "response-status": "403"
  }
}
```

2.4.7. MI.SyntheticResponse

The 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.

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 specify 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: HTTP response status code.

-Type: Integer. Either a static integer or an expression using CDNI-MEL that evaluates to an integer 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: Flag to signal whether the response-status is a static integer or a CDNI-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: body

-Description: 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 CDNI-MEL 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: Flag to signal whether the synthetic response body is a static string literal or a CDNI-MEL expression 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.

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:

```
{
  "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": false
}
```

2.4.8. MI.HeaderTransform

The HeaderTransform object specifies how HTTP headers MUST be added, replaced, or deleted from HTTP requests and responses.

Property: add

-Description: 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

Property: replace

-Description: 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: delete

-Description: 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

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:

```
{
  "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"
  ]
}
```

2.4.9. MI.HTTPHeader

The HTTPHeader object contains a name/value pair for an HTTP header to add or replace in a request or response. The CDNI-MEL expression language can be used to dynamically generate response values.

Property: name

-Description: Name of the HTTP header.

-Type: String

-Mandatory-to-Specify: Yes

Property: value

-Description: New value of the named HTTP header.

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

-Mandatory-to-Specify: Yes

Property: value-is-expression

-Description: Flag to signal whether the value is a static string literal or a [CDNI-MEL] expression 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.

Example of an HTTPHeader illustrating a dynamically constructed header value that equals the session parameter from the query string:

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

2.5. General Metadata

This section documents a set of general purpose GenericMetadata objects whose use and interpretation may be specific to a CDN or Open Caching system's implementation, enabling extensibility and service differentiation for providers.

2.5.1. MI.ServiceIDs

CDN configurations typically have multiple tiers of identifiers that group configurations by customer account to facilitate logging, billing, and support operations. This structure supports two tiers of identifiers (a serviceID which typically identifies a high level customer's service, and a propertyID which typically represents a logical grouping of a set of hosts within a customers' service. It should be noted, however, that the interpretation of ServiceID and PropertyID are implementation-specific, and may not be used by all CDNs and Open Caching systems.

This metadata model extension allows for the association service identifier metadata to a host or path match and to allow for these IDs to be dynamically generated via an expression language. For example, it may be necessary to extract a portion of the Request URI path to derive a service identifier (e.g.: /news/* maps to one propertyID and /movies/* maps to a different propertyID). When processing the MI.ServiceIDs metadata for a request, implementations SHOULD override any previously assigned service identifiers with those specified by this metadata.

MI.ServiceIDs is a new GenericMetadata object that allows for the specification of the two tiers of CDN-specific service identifiers and service names.

Property: service-id

- Description: A provider-specific identifier for the service (typically a customer account identifier).

- Type: String. Either a literal (static string) or an expression using CDNI-MEL to dynamically construct the ID from elements of the HTTP request and/or response.

- Mandatory-to-Specify: No

Property: service-id-is-expression

- Description: Flag to signal whether the service-id is a static string literal or a CDNI-MEL expression that needs to be dynamically evaluated.

- Type: Boolean

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

Property: service-name

- Description: Human-readable name for the service-id.

- Type: String

- Mandatory-to-Specify: No

Property: property-id

- Description: A provider-specific identifier for the property (typically identifies a child configuration within the parent service-id).

- Type: String. Either a literal (static string) or an expression using CDNI-MEL to dynamically construct the ID from elements of the HTTP request and/or response.

- Mandatory-to-Specify: No

Property: property-id-is-expression

-Description: Flag to signal whether the property-id is a static string literal or a CDNI-MEL expression that needs to be dynamically evaluated.

-Type: Boolean

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

Property: property-name

-Description: Human-readable name for the property-id.

-Type: String

-Mandatory-to-Specify: No

Example illustrating the assignment of a literal service-id along with a dynamically computed property-id that is extracted from the root element of the request URI path.

```
{
  "generic-metadata-type": "MI.ServiceIDs",
  "generic-metadata-value": {
    "service-id": "12345",
    "service-name": "My Streaming Service",
    "property-id": "path_element(req.uri, 1)",
    "property-id-is-expression": true
  }
}
```

2.5.2. MI.PrivateFeatureList

The dCDN may gather a certain number of private features (i.e., not [yet] adopted by SVA or considered marginal) that it may want to expose to the content provider and/or the uCDN. Although private, the announcement, selection, and configuration of this private feature could be done through the OC API.

One example could be the support in OCNs of a new protocol that allows the ability to get additional insight about the user agent status (e.g., CTA Wave CMCD).

As another example, Broadpeak has developed a feature called S4Streaming, and would like to give the opportunity to control that feature to the uCDN.

PrivateFeatureListis a GenericMetadata configuration object as a base generic object that permits the control of private features.

Property: features

-Description: The list of feature configuration objects.

-Type: List (array) of MI.PrivateFeature objects .

-Mandatory-to-Specify: Yes

2.5.2.1. MI.PrivateFeature

MI.PrivateFeature object contains the following properties:

Property: feature-oid

-Description: The owner/organization that has specified that feature.

-Type: String

-Mandatory-to-Specify: Yes

Property: feature-type

-Description: Indicates the type/name of the private feature configuration object.

-Type: String

-Mandatory-to-Specify: Yes

Property: feature-value

-Description: Feature configuration object.

-Type: Format/type is defined by the value of the feature-type property above.

-Mandatory-to-Specify: Yes

Note that the private features exposed by the dCDN can be advertised through a dedicated FCI object.

Example, illustrating the Broadpeak S4 Streaming feature:

```
{
  "generic-metadata-type": "MI.PrivateFeatureList",
  "generic-metadata-value": {
    "feature": {
      "feature-oid": "Broadpeak",
      "feature-type": "S4Streaming",
      "feature-value": {
        "footprint": {
          "footprint-type": "ipv4cidr",
          "footprint-value": [
            "192.0.2.0/24",
            "198.51.100.0/24"
          ]
        },
        "activation": "ON",
        "mode": "transparent",
        "policy": "bandwidth-max"
      }
    }
  }
}
```

2.5.3. MI.RequestRouting

The uCDN requires the ability to indicate whether HTTP redirect, DNS redirect, and manifest rewrite are allowed, and indicate which is preferable. This is required in cases where the uCDN would like to delegate the traffic relying on the iterative method but knows the client will not support HTTP redirect. In that case, the uCDN needs a means to force the dCDN to perform request routing based on DNS redirect (or manifest rewrite).

This configuration possibility is useful only if the dCDN can advertise the mode of redirection it supports. There is an ongoing discussion in the IETF CDNI group to understand the semantics behind the redirection modes currently in Footprint & Capabilities Advertising Interface (I-DNS and I-HTTP). It is not clear whether this indicates that the dCDN supports one or both delegation modes (the request routing performed by the uCDN can only be based on DNS redirect or HTTP redirect or both), or whether it indicates that the dCDN supports, as its own request routing mode, DNS redirect and/or HTTP redirect. The latter is required for this new configuration object to be valid.

MI.RequestRouting is a new GenericMetadata object that allows the uCDN to force the dCDN request routing mode(s) to be applied when working in iterative redirection mode. The list of redirection modes supported by the dCDN is advertised through the FCI.RedirectionMode

object. The list of request routing modes supported by the dCDN is advertised through the FCI.RequestRoutingMode object.

Property: request-routing-modes

-Description: Instructs the dCDN to perform request routing according to one or more preferred modes among those supported and advertised by the dCDN through the FCI.RequestRouting object. One must understand that forcing (instead of letting the dCDN request router select) one particular request routing mode may trigger some inefficiency in the request routing process.

-Type: List (array) of iterative request routing modes

-Values: "DNS", "HTTP", "MANIFEST_REWRITE"

-Mandatory-to-Specify: No. By default, all request routing modes supported by the dCDN can be used by the dCDN as part of its request routing process.

Example, illustrating the uCDN forcing the dCDN to use DNS or HTTP as the method for request routing in case the uCDN performs an iterative delegation (i.e., iterative redirection mode):

```
{
  "generic-metadata-type": "MI.RequestRouting",
  "generic-metadata-value": {
    "request-routing-modes": [ "DNS", "HTTP" ]
  }
}
```

3. Metadata Expression Language

The CDNI Metadata Expression Language provides a syntax with a rich set of variables, operators, and built-in functions to facilitate use cases within the extended CDNi metadata model.

Enables expression matching to dynamically determine if StageMetadata (Section 2.4) should be applied at a StageRules match.

Enables the dynamic construction of a value to be used in scenarios such as constructing a service identifier or cache key, setting an HTTP header, rewriting a request URI, setting a response status code, or dynamically generating a response body for a SyntheticResponse.

Expressions can evaluate to a Boolean, 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 as 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
ComputedCacheKey.expression	Dynamically constructs a cache key.	String
ServiceIDs.properry-id,ServiceIDs.service-id	Dynamically constructs service and property identifiers.	String

Table 2: CDNI MEL expressions

3.1. Expression Variables

Variable	Meaning
req.h.<name>	Request header <name>
req.uri	Request URI (includes query string and fragment identifier, if any)

Variable	Meaning
req.uri.path	Request URI path
req.uri.pathquery	Request path and query string
req.uri.query	Request query string
req.uri.query.<key>	Request query string value associated with <key>
req.method	Request HTTP method (GET, POST, others)
resp.h.<name>	Response header <name>
resp.status	Response status code

Table 3: CDNI MEL variables

3.2. Expression Operators and keywords

Operator	Type	Result Type	Meaning
==	infix	Boolean	Equality test
!=	infix	Boolean	Inequality test
!	infix	Boolean	Logical NOT operator
>	infix	Boolean	Greater than test
<	infix	Boolean	Less than test
>=	infix	Boolean	Greater than or equal test
<=	infix	Boolean	Less than or equal
*=	infix	Boolean	Glob style match
~=	infix	Boolean	Regular expression match (see https://www.pcre.org/ for details on PCRE RegEx matching)
ipmatch	infix	Boolean	Match against IP address or CIDR (IPv4 and IPv6)
+	infix	Numeric	Addition
-	infix	Numeric	Subtraction
*	infix	Numeric	Multiplication
/	infix	Numeric	Division
%	infix	Unsigned or Integer	Modulus
.	infix	String	Concatenation
? :	ternary	*	Conditional operator: <e> ? <v1> : <v2> Evaluates <v1> if <e> is true, <v2> otherwise.
( )	grouping		Used to override precedence and for function calls.

Table 4: CDNI MEL expression operators

Keyword	Meaning
and	Logical AND
or	Logical OR
not	Logical NOT (see also the ! operator)

Keyword	Meaning
nil	No value (distinct from empty value)
true	Boolean constant: true
false	Boolean constant: false

Table 5: CDNI MEL expression keywords

3.3. Expression Built-in Functions

3.3.1. Basic Functions: Type Conversions

Function	Action	Argument(s)	Returns
integer(e)	Converts expression to integer.	1	integer
real(e)	Converts expression to real.	1	real
string(e)	Converts expression to string.	1	string
boolean(e)	Converts expression to Boolean.	1	Boolean

Table 6: CDNI MEL type conversions

3.3.2. Basic Functions: String Conversions

Function	Action	Argument(s)	Returns
upper(e)	Converts a string to uppercase. Useful for case-insensitive comparisons.	1	string
lower(e)	Converts a string to lowercase. Useful for case-insensitive comparisons.	1	string

Table 7: CDNI MEL string conversions

3.3.3. Convenience Functions

Function	Action	Argument(s)	Returns
match(string Input, string Match)	Regular expression Match is applied to Input and the matching element (if any) is returned. Empty string is returned if there is no match. See https://www.pcre.org/ for details on PCRE RegEx matching.	2	string
match_replace(string Input, string Match, string Replace)	Regular expression Match is applied to Input arg and replaced with the Replace arg upon successful match.	3	string

Function	Action	Argument(s)	Returns
	Returns updated (replaced) version of Input.		
add_query(string Input, string q, string v)	Add query string element q with value v to the Input string. If v is nil, then just add the query string element q. The query element q and value v must conform to the format defined in: https://datatracker.ietf.org/doc/html/rfc3986	2	string
remove_query(string Input, string q)	Remove (all occurrences of) query string element q from the Input string.	2	string
path_element(string Input, integer n)	Return the path element n from Input. -1 returns the last element.	2	string
path_element(string Input, integer n, integer m)	Return the path elements from position n to m.	3	string

Table 8: CDNI MEL convenience functions

3.4. Error Handling

3.4.1. Compile Time Errors

To ensure reliable service, all CDNI Metadata configurations MUST be validated for syntax errors before they are ingested into a dCDN. That is, existing configurations should be kept as the live running configuration until the new configuration has passed validation. If errors are detected in a new configuration, the configuration MUST be rejected. A HTTP 500 Internal Server Error should be returned with a message that indicates the source of the error (line number, and configuration element that caused the error).

Examples of compile-time errors:

1. Configuration does not parse relative to the CDNI Metadata JSON schema

2. Unknown CDNI Metadata object referenced in the configuration

3. CDNI Metadata object parse error

    a. Missing mandatory CDNI Metadata property

    b. Unknown CDNI Metadata property

    c. Incorrect type for a CDNI Metadata property value

4. CDNI-MEL

    a. Unknown CDNI-MEL variable name referenced in an expression

    b. Unknown CDNI-MEL operator, key-word, or functions referenced in an expression

    c. Incorrect number of arguments used in a CDNI-MEL expression operator or function

    d. Incorrect type of argument used in a CDNI-MEL expression operator or function

3.4.2. Runtime Errors

If a runtime error is detected when processing a request, the request should be terminated, and a HTTP 500 'Internal Server Error' returned to the caller. To avoid security leaks, sensitive information MUST be removed from the error message before it is returned to an external client. In addition to returning the HTTP 500 error, the dCDN SHOULD log additional diagnostics information to assist in troubleshooting.

Examples of runtime errors:

1. Failure to allocate memory (or other server resources) when evaluating a CDNI-MEL expression

2. Incorrect runtime argument type in a CDNI-MEL expression. E.g., trying to convert a non-numeric string to a number

3.5. Expression Examples

3.5.1. ComputedCacheKey

Sets the MI.ComputedCacheKey to the value of the X-Cache-Key header from the client request.

```
{
  "generic-metadata-type": "MI.ComputedCacheKey",
  "generic-metadata-value": {
    "expression": "req.h.x-cache-key"
  }
}
```

Sets the MI.ComputedCacheKey to the lowercase version of the URI.

```
{
  "generic-metadata-type":  "MI.ComputedCacheKey",
  "generic-metadata-value": {
    "expression": "lower(req.uri)"
  }
}
```

3.5.2. ExpressionMatch

ExpressionMatch where the expression is true if the user-agent (glob) matches *Safari* and the referrer equals www.example.com.

```
{
  "expression": "req.h.user-agent *= '*Safari*'
    and req.h.referrer == 'www.example.com'"
}
```

3.5.3. ResponseTransform

Adds X-custom-response-header with a value equal to the value of user-agent - host header.

```
{
  "response-transform": {
    "headers": {
      "add": [
        {
          "name": "X-custom-response-header",
          "value": "req.h.user-agent . ' - ' . req.h.host",
          "value-is-expression": true
        }
      ],
      "response-status": "403"
  }
  }
}
```

Adds a Set-Cookie header with a dynamically computed cookie value (concatenating user agent and host name) and forces a 403 response.

```
{
  "response-transform":{
    "headers":{
      "add":[
      {
      "name":"Set-Cookie",
      "value":"req.h.user-agent . ' - ' . req.h.host",
      "value-is-expression":true
      }
      ]
    }
  }
}
```

3.5.4. MI.ServiceIDs

Extracts the first path element from the URI. For example, if the URI = /789/second/third/test.txt, property-id is set to the first-path (789).

```
{
  "generic-metadata-type":"MI.ServiceIDs",
  "generic-metadata-value":{
    "service-id":"12345",
    "service-name":"My Streaming Service",
    "property-id":"path_element(req.uri, 1)",
    "property-id-is-expression":true
  }
}
```

4. CDNI Capabilities Extensions

Since not all dCDNs will be capable of supporting all the extensions proposed in this document, they need the ability to inform uCDNs about their capabilities. [RFC8008] (the CDNI Footprint & Capabilities Interface) was designed for this purpose and is extended here to express these new capabilities.

4.1. FCI Metadata Object

Whenever a capability is represented as a top-level GenericMetadata object, a dCDN will be able to declare its support simply by including that object name in the capability-value list of the standard FCI.Metadata object.

For each of the new GenericMetadata objects documented within the SVA Configuration Interface, the default assumption should be that the capability is not supported by the dCDN unless named within the FCI metadata object.

Example: A capabilities object declaring support for several of the newly defined GenericMetadata types:

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": [
          "MI.SourceMetadataExtended",
          "MI.ProcessingStages",
          "MI.CrossoriginPolicy",
          "MI.CachePolicy",
          "MI.NegativeCachePolicy",
          "MI.PrivateFeatureList",
          "MI.RequestRouting"
        ]
      },
      "footprints": [
         < Footprint Objects >
      ]
    }
  ]
}
```

4.2. FCI Model Extensions

In most cases, the presence or absence of a GenericMetadata object name in FCI.Metadata (as described above), is sufficient to convey support for a capability. There are cases, however, where more fine-grained capabilities declarations are required. Specifically, a dCDN may support some, but not all, of the capabilities specified by one of the new GenericMetadata objects. In these cases, new FCI objects will be created to allow a dCDN to express these fine-grained capabilities.

4.2.1. FCI.AuthTypes

The AuthTypes object is used to indicate the support of authentication methods to be used for content acquisition (while interacting with an origin server) and authorization methods to be used for content delivery.

This specification document defines two new authentication methods (see MI.Auth) while there is one authorization method currently under specification in CDNI called [URI.signing]

Property: stage-metadata

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

-Type: Array of StageMetadata objects, applied in order.

-Mandatory-to-Specify: Yes

Property: authe-types

-Description: List of supported authentication methods (possibly required for content acquisition)

-Type: Array of strings

-Values: "AWSv4Auth", "HeaderAuth"

-Mandatory-to-Specify: No. No authentication method is supported in this case.

Property: autho-types

-Description: List of supported authorization methods (possibly required for content delivery)

-Type: Array of strings

-Values: "UriSigning"

-Mandatory-to-Specify: No. No authorization method is supported in this case.

FCI.AuthTypes example:

```
{
  "capabilities": [
    {
      "capability-type": "FCI.AuthTypes",
      "capability-value": {
        "authe-types": [
          "AWSv4Auth",
          "HeaderAuth"
        ],
        "autho-types": [
          "UriSigning"
        ]
      }
    }
  ]
}
```

4.2.2. FCI.ProcessingStages

This object is used to signal the set of features that are supported in relation to the ProcessingStages configuration object (see MI.ProcessingStages). Those optional features depend on the CDNI-MEL language support.

Property: features

-Description: List of supported optional processing stages features. Note that these features all have some dependencies on support of the CDNI MEL expression language.

-Type: Array of strings

-Values: "ExpressionMatch", "RequestTransform", "ResponseTransform"

-Mandatory-to-Specify: No. None of these optional features are supported in this case.

Example:

```
{
  "capabilities": [
    {
       "capability-type": "FCI.ProcessingStages",
       "capability-value": {
         "features": [
           "ExpressionMatch",
           "RequestTransform",
           "ResponseTransform"
         ]
       }
     }
  ]
}
```

4.2.3. FCI.SourceMetadataExtended

This object is used to signal the supported features related to the SourceMetadataExtended configuration object.

Property: load-balance

-Description: List of supported load balancing algorithms in relation to the SourceMetadataExtended configuration object (see MI.SourceMetadataExtended)

-Type: Array of strings

-Values: "random", "content-hash", "ip-hash

-Mandatory-to-Specify: No. load balancing is not supported among sources.

If the FCI.SourceMetadtaExtended object is not exposed/advertised or if the "load-balance" array is empty, the dCDN does not support the usage of the load-balance property attached to the SourceMetadataExtended configuration object (see MI.SourceMetadataExtended).

Example:

```
{
  "capabilities": [
    {
      "capability-type": "FCI.SourceMetadataExtended",
      "capability-value": {
        "load-balance": [
          "random",
          "content-hash",
          "ip-hash"
        ]
      }
    }
  ]
}
```

4.2.4. FCI.RequestRouting

This object is used by the dCDN to signal/announce the supported request routing modes. This can be optionally used by the uCDN to further select a subset of those modes when operating one of the iterative delegation modes. See the section on the GenericMetadata RequestRouting object..

Property: request-routing-modes

-Description: List of supported request routing modes by the dCDN. This information is useful when the uCDN decides to perform a delegation in iterative mode.

-Type: Array of strings

-Values: "DNS", "HTTP-R", "MANIFEST_REWRITE"

-Mandatory-to-Specify: No. If the dCDN does not advertise the supported request routing modes, they are all supported by default.

Example:

```
{
  "capabilities": [
    {
      "capability-type": "FCI.RequestRouting",
      "capability-value": {
        "request-routing-modes": [
          "DNS",
          "HTTP",
          "MANIFEST_REWRITE"
        ]
      }
    }
  ]
}
```

4.2.5. FCI.PrivateFeatures

This object is used by the dCDN to signal/announce the list of supported private features. See the section on the GenericMetadata PrivateFeatureList object.

Property: features

-Description: The list of supported private feature

-Type: List nested objects of FCI.PrivateFeature

Example:

```
{
  "capabilities": [
    {
      "capability-type": "FCI.PrivateFeatures",
      "capability-value": {
        "features": [
          {
            "feature-oid": "Broadpeak",
            "feature-type": "S4Streaming"
          }
        ]
      }
    }
  ]
}
```

4.2.5.1. FCI.PrivateFeature

This object contains the following properties:

Property: feature-oid

-Description: The owner/organization that has specified the feature.

-Type: String

-Mandatory-to-Specify: Yes

Property: feature-type

-Description: Indicates the type/name of the private feature configuration object.

-Type: String

-Mandatory-to-Specify: Yes

4.2.6. FCI.OcnSelection

This object is used by the dCDN to signal/announce the supported OCN types and/or their transport arrangement and/or medium supported by OCNs.

Property ocn-delivery-list

1. Description: List of supported medium and/or transport arrangements.

2. Type: Array of nested objects, each containing the following properties:

o Property: ocn-medium

-Description: This property lists the supported mediums.

-Type: Array of strings. The following values are specified: "SATELLITE"

-Mandatory-to-Specify: No

o Property: ocn-transport

-Description: Instructs the dCDN to perform delegation operating a particular transport arrangement. The following values are specified: "MABR".

-Type: Array of strings

-Mandatory-to-Specify: No

.....Property: ocn-type-list

o Description: List of supported OCN types. Examples include: "HOME" or "EDGE".

o Type: Array of strings

o Mandatory-to-Specify: No

5. IANA Considerations

5.1. 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
MI.AllowCompress	RFCthis
MI.SourceMetadataExtended	RFCthis
MI.SourceExtended	RFCthis
MI.LoadBalanceMetadata	RFCthis
MI.HeaderAuth	RFCthis
MI.AWSv4Auth	RFCthis
MI.CrossOriginPolicy	RFCthis
MI.AuthTokenMetadata (TBD)	RFCthis
MI.CertificateMetadata (TBD)	RFCthis
MI.OcnSelection	RFCthis
MI.RequestRouting	RFCthis
MI.ProcessingStages	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
MI.ServiceIDs	RFCthis

Payload type	Specification
MI.TrafficType	RFCthis
MI.LoggingMetadata (TBD)	RFCthis
MI.PrivateFeatureList	RFCthis
FCI.AuthTypes	RFCthis
FCI.ProcessingStages	RFCthis
FCI.SourceMetadataExtended	RFCthis
FCI.RequestRouting	RFCthis
FCI.PrivateFeatures	RFCthis
FCI.OcnSelection	RFCthis

Table 9: Payload Types

6. Security Considerations

This specification is in accordance with the CDNI Request Routing: Footprint and Capabilities Semantics. As such, it is subject to the security and privacy considerations as defined in Section 8 of [RFC8006] and in Section 7 of [RFC8008] respectively.

7. Conclusion

This document presents requirements and extensions to the CDNI metadata model to cover typical use cases found in the commercial CDN and Open Caching ecosystems. By limiting the scope of these extensions to new GenericMetadata objects, backward compatibility can be maintained with any existing CDNI Metadata Interface implementations.

8. References

8.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC

7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[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>.

[RFC8007] Murray, R. and B. Niven-Jenkins, "Content Delivery Network Interconnection (CDNI) Control Interface / Triggers", RFC 8007, DOI 10.17487/RFC8007, December 2016, <https://www.rfc-editor.org/info/rfc8007>.

[RFC8008] Seedorf, J., Peterson, J., Previdi, S., van Brandenburg, R., and K. Ma, "Content Delivery Network Interconnection (CDNI) Request Routing: Footprint and Capabilities Semantics", RFC 8008, DOI 10.17487/RFC8008, December 2016, <https://www.rfc-editor.org/info/rfc8008>.

[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>.

[RFC8804] Finkelman, O. and S. Mishra, "Content Delivery Network Interconnection (CDNI) Request Routing Extensions", RFC 8804, DOI 10.17487/RFC8804, September 2020, <https://www.rfc-editor.org/info/rfc8804>.

[URI.signing] van Brandenburg, R., Leung, K., and P. Sorber, "URI Signing for CDN Interconnection (CDNI)", 8 October 2019, <http://www.ietf.org/internet-drafts/draft-ietf-cdni-uri-signing-19.txt>.

[W3C] "Cross-Origin Resource Sharing", <https://www.w3.org/TR/2020/SPSD-cors-20200602/>.

8.2. Informative References

[AWSv4Method] "Authenticating Requests (AWS Signature Version 4)", <https://docs.aws.amazon.com/AmazonS3/latest/API/sig-v4-authenticating-requests.html>.

[OC-CI] Goldstein, G., Ed., Power, W., Bichot, G., and A. Siloniz, "Open Caching - Configuration Interface Functional Specification (Parts 1,2,3)", Version 0.1, 2 July 2021.

[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>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <https://www.rfc-editor.org/info/rfc6707>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <https://www.rfc-editor.org/info/rfc7336>.

[RFC7694] Reschke, J., "Hypertext Transfer Protocol (HTTP) Client-Initiated Content-Encoding", RFC 7694, DOI 10.17487/RFC7694, November 2015, <https://www.rfc-editor.org/info/rfc7694>.

[SVA] "Streaming Video Alliance Home Page", <https://www.streamingvideoalliance.org>.

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

Guillaume Bichot
Broadpeak
France

Email: guillaume.bichot@gmail.com

Alfonso Siloniz
Telefonica
Spain

Email: alfonsosiloniz@gmail.com