Network Working Group
Internet-Draft
Obsoletes: 8007 (if approved)
Intended status: Standards Track
Expires: 23 April 2026

N.B. Sopher
O. Finkelman
Qwilt
S. Mishra
Verizon
J.K. Robertson
Qwilt
A. Arolovitch
Viasat

20 October 2025

# Content Delivery Network Interconnection (CDNI) Control Interface / Triggers 2nd Edition
## draft-ietf-cdni-ci-triggers-rfc8007bis-18

## Abstract

This document obsoletes RFC8007. The document describes the part of Content Delivery Network Interconnection (CDNI) Control interface that allows a CDN to trigger activity in an interconnected CDN that is configured to deliver content on its behalf. The upstream CDN MAY use this mechanism to request that the downstream CDN preposition, invalidate, and/or purge metadata and/or content. The upstream CDN MAY monitor the status of activity that it has triggered in the downstream CDN.

## 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 23 April 2026.

Table of Contents

## 1. Introduction

[RFC6707] introduces the problem scope for Content Delivery Network Interconnection (CDNI) and lists the four categories of interfaces that may be used to compose a CDNI solution (Control, Metadata, Request Routing, and Logging).

[RFC7336] expands on the information provided in [RFC6707] and describes each of the interfaces and the relationships between them in more detail.

The CDNI Control Interface / Triggers 1st edition [RFC8007], deprecated by this document, describes the "CI/T" interface -- "CDNI Control Interface / Triggers". It does not consider those parts of the Control interface that relate to the configuration, bootstrapping, or authentication of CDN Interconnect interfaces. Section 4 of [RFC7337] identifies the requirements specific to the CI/T interface; requirements applicable to the CI/T interface are CI-1 to CI-6.

This document is a second edition of the CDNI Control Interface / Triggers, which defines a new version, "v2", of the interface objects. The new version aims to support REST [REST] architectural style in a way that improves the interface’s flexibility, extensibility, and interoperability, and allows encoding of the interface using OpenAPI. The new objects replace the main CI/T objects as follows:

* The "ci-trigger-command" object and its matching "ci-trigger-status" object are replaced with the "ci-trigger.v2" object representing a trigger resource

* The "ci-trigger-collection" object is replaced with the "ci-trigger-collection.v2" that is expanded to support filtering by trigger state and trigger labels

The second edition of the CI/T interface further allows the use of separate Control interface endpoints for content and metadata.

The document also provides a trigger extension mechanism that MAY be used to provide further instruction on the trigger execution.

This second edition also includes cascaded CDN error propagation and extended trigger status reporting for improved trigger execution monitoring, as well as the use of external object lists for improved scale and integration of trigger-based APIs with existing content workflows.

* Section 2 outlines the model for the CI/T interface at a high level.

* Section 3 defines the CI/T interface offered by the downstream CDN.

* Section 4 defines the encoding of the standard CI/T objects and introduces trigger spec and trigger extension types.

* Section 5 describes the FCI capabilities objects used to inform on the supported CI/T-related capabilities.

* Section 6 contains example messages.

### 1.1. Terminology

This document reuses the terminology defined in [RFC6707] and uses "uCDN" and "dCDN" as shorthand for "upstream CDN" and "downstream CDN", respectively.

Additionally, the following terms are used throughout this document and are defined as follows:

* HLS - HTTP Live Streaming

* DASH - Dynamic Adaptive Streaming Over HTTP

* MSS - Microsoft Smooth Streaming

The keywords "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. Model for CDNI Triggers

### 2.1. REST Architecture

The CI/T interface utilizes the HTTP/1.1 protocol [RFC9112] and follows the principles of the Representational State Transfer (REST) architectural style. The uCDN, in its capacity as a CI/T interface client, requests the dCDN to carry out an action ("trigger") related to metadata or content stored by the dCDN on behalf of the uCDN.

The dCDN, as a CI/T interface server, governs the triggers as a set of resources, which can be dynamically created and deleted, and whose state can be retrieved and/or modified by the uCDN. Each such trigger is identified by a unique Uniform Resource Identifier (URI) as defined in Section 4.2 of [RFC9110].

Once a trigger is created, the uCDN can retrieve its representation from the dCDN or request the trigger to be modified by transferring an updated representation of it to the dCDN. The CI/T interface supports the representation of trigger resources using JSON [RFC8259].

This RESTful data model, built around a common "trigger" resource, replaces the command-oriented model of [RFC8007], wherein the uCDN passed commands to the dCDN using "ci-trigger-command" objects, and the dCDN generated "ci-trigger-status" objects in response.

## 2.2. HTTP Methods

Section 9.3 of [RFC9110] defines the set of methods in HTTP. The CI/T interface uses some of these methods for resource creation, retrieval of resource state, modification of resources, and deletion of resources. The HTTP methods not listed here are not supported by the CI/T interface.

* GET - used to retrieve the current state of a resource. The GET method doesn’t cause any state change on the server side.

* POST - used to request that the target resource process the representation enclosed in the request. If a resource has been created on the server as a result of successfully processing a POST request, the server sends a 201 ("Created") response containing a Location header field that contains an identifier for the newly created resource.

* DELETE - used to request the server remove the target resource.

* HEAD - used to request metadata associated with the target resource, in the form of HTTP response headers that would have been sent if the GET method were used instead. The HEAD method can be used to verify that the target resource exists on the server.

## 2.3. Trigger

The uCDN requests creation of a trigger resource to instruct the dCDN to perform an action. If the dCDN accepts the request, it creates a new trigger resource and returns its unique URI to the uCDN. The uCDN MUST use this URI for all requests associated with the created trigger resource.

Note that the version of the trigger resources that the uCDN requests to create MUST match the version of CI/T trigger objects reported as supported by the dCDN.

The CI/T interface supports the following types of trigger action:

* preposition - used to instruct the dCDN to fetch metadata from the uCDN or content from any origin, including the uCDN.

* invalidate - used to instruct the dCDN to revalidate specific metadata or content before reusing it.

* purge - used to instruct the dCDN to delete specific metadata or content.

Note that additional action types can be defined and registered in the future.

The trigger resource has a "state" attribute. The dCDN creates new triggers in the "pending" state. Once the dCDN starts processing a pending trigger, the trigger state is set to "active". The uCDN MAY explicitly request the trigger to be created in the "active" state. If accepted by the dCDN, it may create the new trigger in the "active" state and start its processing immediately upon creation. Once the trigger processing is complete, the state is set to either "complete" or "failed", depending on the processing outcome.

The uCDN MAY request the cancellation of a trigger from the dCDN. If such a request is accepted, the trigger state is changed to "cancelling", and when the cancellation is complete, the trigger state changes to "cancelled".

For a full description of the trigger resource, please refer to Section 4.1.

## 2.4. Trigger Access Control and Multi-Tenancy

The dCDN MUST only allow the uCDN access to the trigger resources it created.

The dCDN MUST ensure that triggers created in response to a uCDN request apply only to content and metadata objects associated with that uCDN.

In case of content prepositioning, the dCDN MUST be able to associate content objects referenced in a trigger created by the uCDN with delivery CDNI metadata objects in its possession that are associated with the same uCDN. These CDNI metadata objects include HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch, and PathMetadata, as described in Section 3.1 of [RFC8006]. For example, if the dCDN has no MI metadata objects that enable the dCDN to respond to requests for video.example.com, it MUST NOT allow prepositioning of content objects with this hostname in the object URL.

In case of metadata prepositioning, the prepositioned metadata objects MUST be consistent with pre-existing metadata, e.g., prepositioning of the MI HostMatch object in the absence of MI HostIndex would be rejected.

If such association between a trigger and pre-existing delivery metadata cannot be established, the dCDN MUST reject it.

Furthermore, the dCDN SHOULD reject a trigger from the uCDN A that seeks to preposition delivery metadata objects that are in conflict with the pre-existing metadata objects belonging to another uCDN B, e.g., if it could cause the dCDN to match a content request with metadata objects from multiple uCDNs.

## 2.5. Trigger Index and Trigger Collections

The Trigger Index is the top-level resource that references all trigger resources belonging to a particular uCDN. The dCDN maintains one Trigger Index resource for each uCDN, and each uCDN has access only to its own Trigger Index. The Trigger Index contains references to Trigger Collection resources. Each collection contains triggers that are optionally filtered by parameters - for example, all triggers in the "pending" state or all triggers labeled "video". The same trigger resource can be present in more than one collection. The supported trigger collection representations are listed in Section 4.3 and include filtering of triggers by state and label. Note that additional trigger collection representations can be defined in the future.

## 2.6. Session Overview

Figure 1 is an example showing the basic message flow in a CI/T interface session used by the uCDN to trigger activity in the dCDN and for the uCDN to discover the status of that activity. Only successful triggering is shown. Please note that the example below uses simplified trigger identifiers for brevity. It is recommended that the actual implementation uses unique UUID identifiers as specified in [RFC9562]. Examples of the messages are shown in Section 6.

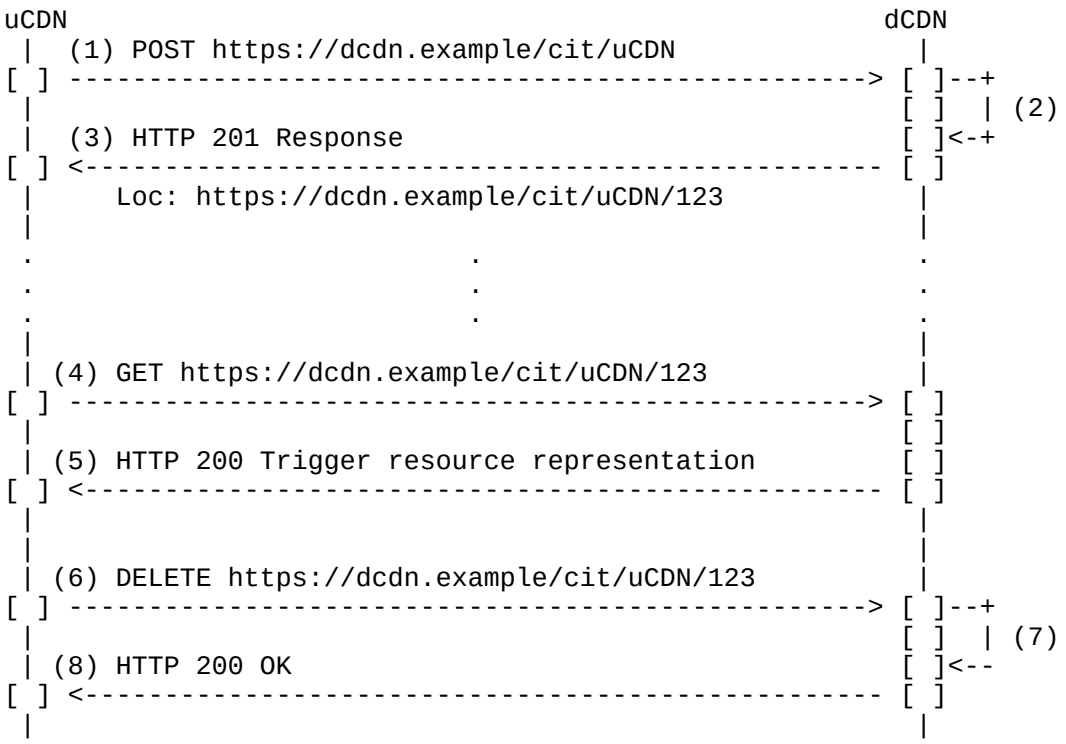



Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are as follows:

1. The uCDN requests creation of a new trigger resource by POSTing its representation to the trigger index resource with a well-known URI "https://dcdn.example/cit/uCDN".

2. The dCDN authenticates the request, validates the trigger resource in it, and if the request is accepted, creates a new trigger resource.

3. The dCDN responds to the uCDN with an HTTP 201 ("Created") response status and the location of the trigger resource.

4. The uCDN MAY query, possibly repeatedly, the trigger resource in the dCDN.

5. The dCDN responds to each query with the current trigger resource representation, including the trigger state, that reflects the progress of the uCDN request.

6. Once the trigger reaches a terminal state ("complete", "processed", "failed" or "cancelled"), the uCDN MAY request deletion of the trigger resource.

7. The dCDN validates the request and the trigger resource state. If successful, the trigger resource is removed by the server, and subsequent requests for this resource MUST result in 404 ("Not Found").

8. The dCDN responds to the deletion request with a 200 ("OK") status code.

This section provides an overview of a regular session. For detailed discussions of trigger modification, cancellation, and deletion, see Section 3.2, Section 3.3, and Section 3.5.

## 2.7. Trigger Processing

### 2.7.1. Timing and Order

The uCDN MAY place limits on the timing and order of execution of a trigger through optional TimePolicy (Section 4.1.3.3.2) and/or ExecutionPolicy (Section 4.1.3.3.3) extensions. If neither of these extensions are present in the trigger resource, the timing and order of the trigger execution is under the dCDN’s control, including the start time, pacing of the activity in the network, and order in which the dCDN chooses to process pending triggers.

The CI/T "invalidate" and "purge" trigger actions MUST be applied to all data acquired before the dCDN begins the trigger processing (i.e., enters "active" state). The dCDN implementation SHOULD apply "invalidate" and "purge" triggers to content acquisition that is in progress when the trigger becomes active, to avoid placing purged or invalidated content into the cache upon completion of the content acquisition. The dCDN SHOULD NOT apply CI/T "invalidate" and "purge" actions to data acquired after the trigger processing started, but this may not always be achievable, so the uCDN cannot count on that.

If the uCDN wishes to invalidate or purge content and then immediately preposition replacement content at the same URLs, it SHOULD ensure that the dCDN has completed the invalidate/purge before initiating the prepositioning. Otherwise, there is a risk that the dCDN prepositions the new content, then immediately invalidates or purges it (as a result of the two uCDN requests running in parallel). The uCDN MAY use the Execution Policy (Section 4.1.3.3.3) extension to condition the start of preposition trigger processing on completion of the earlier invalidate/purge trigger(s).

#### 2.7.2. Scope

Each trigger can operate on multiple metadata and content elements. These elements are targeted by specifying both their subject (i.e., "metadata" or "content") as well as specification method (e.g., URL Regexes) and value.

Multiple representations of an HTTP resource may share the same URL. Triggers that invalidate or purge metadata or content apply to all resource representations with matching URLs.

#### 2.7.3. Results

Possible trigger states are defined in Section 4.1.5.

Trigger state MUST NOT be reported as "complete" until all operations listed in the trigger have been completed successfully. In case of CDN cascading, the completion of operations includes processing of the trigger in downstream CDNs. For detailed discussion of the cascading use case, see Section 2.9. The reasons for failure, and URLs or patterns affected, SHOULD be made available in the trigger state representation. For more details about error handling, see Section 3.7.

### 2.8. Trigger Extensibility

The CDNI Control Interface / Triggers 1st edition [RFC8007] defines a set of properties and objects used by the trigger commands. This 2nd edition defines an extension mechanism to the triggers interface that enables applications to add instructions for finer control over the trigger execution, for example indicating a time window in which to execute the trigger. This document specifies a generic trigger extension object wrapper for managing individual CDNI trigger extensions in an opaque manner.

All trigger extensions are OPTIONAL, and it is thus the responsibility of the extension specification to define a consistent default behavior for the case the extension is not present.

All trigger extensions MUST be registered in the IANA "CDNI CI/T Trigger Extensions" registry (see Section 7.6).

This document also defines an initial set of trigger extension objects and registers them in the IANA "CDNI CI/T Trigger Extensions" registry:

| JSON string | Description |
|---|---|
| location-policy | Allowing the control over the locations in which the trigger is executed. |
| time-policy | Allowing the scheduling of a trigger to run in a specific time window. |
| execution-policy | Allowing the control over the order and timing in which triggers are executed. |

Table 1

## 2.9. Multiple Interconnected CDNs

In a network of interconnected CDNs, a single uCDN will originate a given item of metadata and associated content. It MAY distribute that metadata and content to one or more dCDNs, which in turn distributes that metadata and content to additional dCDNs located further downstream.

A transit CDN is a dCDN that passes on CDNI Metadata and content to the dCDNs located further downstream.

The dCDN that creates trigger resources at the request of such transit CDN MUST associate the triggers with the transit CDN from which it receives the request, regardless of where the trigger request may have originated.

A "diamond" configuration is one where the dCDN can potentially acquire metadata and content originated in one uCDN from that uCDN itself and a transit CDN, or via more than one transit CDN.

The "diamond" configuration may cause configuration consistency problems, where the dCDN may end up in possession of multiple, potentially conflicting metadata objects belonging to the multiple uCDNs that match the same content request. The conflict may arise due to the differences in trigger processing by the transit CDNs and/or variances in trigger propagation time across different paths in the "diamond" topology.

Because of this, the "diamond" configuration is considered a configuration error. A dCDN that receives identical trigger creation requests from different uCDNs SHOULD reject duplicate trigger requests, as described in Section 2.4.

Security considerations are discussed further in Section 8.

If the dCDN is also acting as the uCDN in a cascade, it MUST forward trigger requests to any dCDNs that may be affected. The trigger state MUST NOT be reported as "complete" by a transit CDN until it is "complete" in all of its dCDNs and in the transit CDN itself. If a trigger is reported as "processed" in the transit CDN or any one of its dCDNs, transit CDNs MUST report the trigger as "processed" as well. If a trigger is reported as "failed" by the transit CDN or any one of its dCDNs, the transit CDN must report the trigger as "failed" only after its processing is finished in it and all of its dCDNs. A cancelled trigger MUST be reported as "cancelling" until it has been reported as "cancelled", "complete", or "failed" by all cascaded dCDNs.

## 2.10. Loop Detection and Prevention

Given three CDNs, A, B, and C, if CDNs B and C delegate delivery of CDN A’s content to each other, CDN A’s trigger creation requests could be passed between CDNs B and C in a loop. More complex networks of CDNs could contain similar loops involving more hops.

When such CDN topologies become possible, it is RECOMMENDED that CDNs participating in it utilize a CDN Provider ID (PID) (Section 4.4.3) to detect and prevent loops as follows:

* The uCDNs that originate a new trigger request SHOULD specify their CDN provider ID using the trigger "cdn-path" attribute (see Section 4.1 for details).

* A dCDN that receives a trigger creation request that contains a "cdn-path" attribute SHOULD check it for its own CDN PID. If the dCDN’s PID is already present, and the dCDN is not the CDN initiating the trigger, this condition likely indicates a loop. In such case, the dCDN MUST reject the trigger, which would result in a trigger rejection being returned to the originating uCDN. If the dCDN receives a trigger that it itself originated, the dCDN MAY process the trigger as required.

* A dCDN that cascades trigger requests to additional dCDNs (so-called "transit CDN") SHOULD NOT reject triggers that have CDN PID of its downstream CDNs in their CDN path, allowing each CDN to do their own loop detection.

* Transit CDNs MUST append their CDN PID to the CDN path of a trigger before sending it to its downstream CDNs.

* The dCDNs SHOULD advertise their CDN provider ID to the uCDNs using the "cdn-id" attribute of the trigger index (Section 4.2) resource.

## 3. CDNI Trigger Interface

This section describes an interface to enable the uCDN to trigger activity in the dCDN.

The CI/T interface builds on top of HTTP, so the dCDNs may make use of any HTTP feature when implementing the CI/T Interface. For example, the dCDN SHOULD make use of HTTP's caching mechanisms to reduce the uCDN's trigger status polling overhead by indicating the modification status of a requested resource representation.

The dCDNs MAY implement separate CI/T interfaces per Section 4.1.2.2, i.e., one CI/T interface for trigger operations on metadata and another for operations on content. In this case, the dCDN MUST advertise separate interface endpoints via Section 5.1.

All dCDNs implementing CI/T MUST support the HTTP GET, HEAD, POST, and DELETE methods as defined in [RFC9110].

The only resource representation specified in this document is JSON [RFC8259]. It MUST be supported by both the uCDN and the dCDN.

The CI/T interface uses a root URI for the retrieval of the trigger index resource and creation of new triggers. The mechanism for discovery of that URL is part of the CI/T interface bootstrapping and is outside the scope of this document.

The uCDN requests to create a new trigger resource by POSTing its representation to the trigger index resource URI, discovered at the time of interface bootstrapping, e.g., "https://dcdn.example/cit/ucdn/triggers". If the request is accepted by the dCDN, it creates a new trigger resource and returns its URI to the uCDN in an HTTP 201 ("Created") response.

Once created, the new trigger URI also becomes available via the trigger collection resources described in Section 4.3. Additionally, the uCDN may discover the URIs of multiple trigger collection representations by retrieving the trigger index resource, which is accessible at the interface root URI. This means that the URIs for all trigger resources and trigger collection representations can be discovered by the uCDN through the top-level trigger index resource, allowing dCDNs to use any URI structure they choose for CI/T resources. Therefore, the uCDNs MUST NOT make any assumptions regarding the structure of CI/T URIs or the mapping between CI/T

objects and their associated URIs. The URIs used in the examples in this document are purely illustrative and are not intended to impose a definitive structure on CI/T interface implementations.

### 3.1. Creating Triggers

To create a new trigger, the uCDN makes an HTTP POST request with the trigger representation to the trigger index resource URI. The trigger representation MUST include the mandatory attributes of the trigger resource (Section 4.1).

The uCDN MAY also optionally specify optional trigger v2 specification attributes, namely trigger labels and trigger extensions, as well as the optional "cdn-path" attribute of the trigger resource.

The dCDN validates the trigger resource representation sent by the uCDN. If the representation is malformed or the uCDN does not have sufficient access rights, the dCDN MUST either respond with an appropriate 4xx HTTP error code and not create a trigger resource or create a trigger resource with a "failed" state and an appropriate Error.v2 Description (Section 4.1.6.1).

The new trigger resource is created in a "pending" state. If successful, The HTTP response to the uCDN trigger creation request MUST have status code 201 ("Created") and MUST convey the URI of the newly created trigger resources in the Location response header field [RFC9110]. The HTTP response SHOULD include the updated representation of the trigger resource. This is particularly important in cases where the dCDN processed the trigger immediately.

Once a trigger resource has been created, the dCDN MUST NOT reuse its URI, even after the trigger resource has been fully removed. It is therefore recommended that the dCDN utilize unique UUID identifiers as specified in [RFC9562].

The dCDN SHOULD provide continuous updates of the trigger processing progress by responding with updated trigger resource representations to the subsequent uCDN requests sent to the created trigger URI. If the dCDN is unable to do that, it MUST indicate that it has accepted the request but will not be providing further status updates. To do this, it MUST set the trigger state to "processed" at creation time. In this case, CI/T processing should continue as if it were a request in the "complete" state. In this case, the dCDN SHOULD also provide an estimated completion time for the request by using the trigger "etime" property.

The uCDN MAY request the new trigger to be created in the "active" state so that its processing begins immediately. If agreed to by the dCDN, the dCDN MUST start the new trigger processing immediately.

Otherwise, the dCDN MUST set the state of the new trigger to "pending". Once trigger processing has started, the status MUST be changed to "active". Finally, once the trigger processing is complete, the trigger state MUST be set to "complete" or "failed".

Once created, trigger resources can be cancelled, modified, or deleted by the uCDN, subject to the constraints described below.

## 3.2. Modifying Triggers

Modification of existing triggers is useful for the uCDN to correct an error in trigger specification or trigger extension(s) that may govern when the trigger is to be processed.

The uCDN can request modification of an existing trigger resource by sending an updated trigger representation to the trigger URI using HTTP POST command.

The dCDN MAY accept modifications of the trigger specifications, trigger extensions and trigger labels, when the trigger is in a "pending" state, i.e., the dCDN didn’t start its processing yet.

The dCDN MAY also accept a request to change the trigger state subject to the following constraints:

* the requested state is "cancelled", and the trigger was in either "pending" or "active" state when the dCDN receives the request

* the requested state is "active", and the trigger was in a "pending" state when dCDN received the request

Section 3.3 describes the processing of the trigger cancellation requests in detail. The uCDN MAY request to set the trigger state to "active" to prompt the dCDN to re-examine the trigger resource and start its processing immediately.

The dCDN MUST respond to the trigger modification request appropriately. Thus, the HTTP status code 200 ("OK") should be returned if the modification has been processed, 202 ("Accepted") if the command has been accepted but the modification is not fully complete yet, 404 ("Not Found") when the trigger resource does not exist, 409 ("Conflict") when the trigger resource is in a state that doesn't allow the requested modification, 501 ("Not Implemented") if the modification is not supported by the dCDN or an appropriate 4xx HTTP error code in case of a malformed request.

In case of successful 2xx response, the dCDN MUST provide the updated trigger resource representation in the response body.

### 3.3. Cancelling Triggers

The uCDN MAY request cancellation of a trigger by requesting its state to be set to "cancelled", as described in Section 3.2. The dCDN MUST respond to such request, however, the actual cancellation of a trigger resource is optional to implement.

The dCDN MUST respond to the trigger cancellation request with an appropriate HTTP response status code as documented in Section 3.2

If cancellation of a "pending" trigger is accepted by the dCDN, the dCDN SHOULD NOT start the processing of that activity. Requesting a cancellation of a "pending" trigger does not, however, guarantee that the corresponding activity will not be started because the uCDN cannot control the timing of that activity. Processing could, for example, start after the POST is sent by the uCDN but before that request is processed by the dCDN.

If cancellation of an "active" or "processed" trigger is accepted by the dCDN, the dCDN SHOULD stop processing the trigger. However, as with the cancellation of a "pending" trigger, the dCDN does not guarantee that the trigger processing doesn't run to completion in the meantime.

If the dCDN cannot stop the trigger processing immediately after receiving the request from the uCDN to do so, it MUST set the trigger state to "cancelling" and provide this state in the trigger representation in its response. If the trigger processing is stopped before its normal completion, the trigger state MUST be set to "cancelled".

Cancellation of a "complete", "failed" or "cancelled" trigger requires no processing in the dCDN. Its state MUST NOT be changed.

### 3.4. Checking Status

The uCDN has two ways to check the progress of its triggers' processing, as described in Section 3.4.1 and Section 3.4.2.

To enable the uCDN to use client-side caching of the trigger index resource, as well as all trigger and trigger collection resources, each resource representation sent by the dCDN SHOULD include at least one of the following HTTP response headers: "ETag" or "Last-Modified". The dCDN should respond with the HTTP 304 ("Not Modified") status code and no response body for conditional resource requests using the 'If-None-Match' and/or 'If-Modified-Since' headers, as specified in Section 13 of [RFC9110], if it does not have a more recent resource representation.

The dCDN SHOULD also use the cache control headers for responses to GET requests for its resources to indicate the frequency at which it recommends that the uCDN and/or intermediate proxies should poll for change. If provided, the uCDN should match the frequency of polling to the cache control information provided by the dCDN.

#### 3.4.1. Polling Trigger Collections

The uCDN MAY fetch the Trigger Collection that contains all of its triggers, or one of the collections that filter triggers based on a parameter such as state or label. This makes it possible for the uCDN to poll the status of all trigger resources or selected trigger subsets.

##### 3.4.1.1. Extended view representation

If the dCDN advertises support for extended status, the uCDN MAY request the extended trigger collection representation, which embeds full representations of trigger resources in the collection resource. The extended representation is supported for all trigger collections, so it is possible to retrieve all trigger resource representations for a specific trigger state (e.g., all triggers in a "pending" state).

The uCDN SHOULD request the extended representation by passing the query string parameter "status=extended" when requesting a trigger collection resource. The dCDN SHOULD respond with HTTP Status 200 ("OK") when such request can be satisfied, 501 ("Not Implemented") if the capability has not been implemented or advertised, and 400 ("Bad Request") for a malformed query.

By default, the trigger resources are represented in the trigger collection with their resource URI only.

#### 3.4.2. Polling Triggers

The uCDN has a URI provided by the dCDN at the trigger creation time. Alternatively, the uCDN MAY discover a trigger resource URI by retrieving the trigger index resource, and then the appropriate trigger collection referenced in the index. It may fetch an up-to-date representation of the trigger resource at any time using an HTTP GET request, including changes in the trigger state, as well as the outcome of the trigger processing.

### 3.5. Deleting Triggers

The uCDN MAY request the deletion of trigger resources at any time using the HTTP DELETE method, as defined in the CDNI Control Interface / Triggers 1st edition [RFC8007],

Once deleted, the deleted trigger MUST be removed from all trigger collections. Subsequent requests to the trigger resource URI MUST be rejected by the dCDN with HTTP error 404 ("Not Found").

The effect of deletion is similar to cancellation, except that the trigger resource becomes unavailable after the deletion is complete. For this reason, the uCDN SHOULD cancel triggers rather than delete them when it needs to access the trigger status after processing has terminated.

If a "pending" trigger is deleted, the dCDN SHOULD NOT start the processing of that activity. Deleting a "pending" trigger does not, however, guarantee that its processing has not started, because the uCDN cannot control the timing of that activity. Processing may, for example, start after the DELETE is sent by the uCDN but before that request is processed by the dCDN.

When an "active" or "processed" trigger is deleted, the dCDN SHOULD stop processing it. However, as with the deletion of a "pending" trigger, the dCDN does not guarantee this.

Deletion of a "complete", "cancelled", "cancelling", or "failed" trigger MUST result in no further processing by the dCDN.

The dCDN MUST respond to the trigger deletion request appropriately. The dCDN MUST respond with status code 200 (OK) without a response body if the trigger has been deleted immediately, 202 (Accepted) if the command has been accepted but the trigger has not yet been deleted, 404 (Not Found) if the trigger resource does not exist, or 501 (Not Implemented) if deletion is not supported by the dCDN.

The trigger state MUST be set to "cancelling" while the dCDN is processing a deletion request asynchronously.

### 3.6. Expiry of Triggers

The dCDN MAY automatically delete trigger resources sometime after they reach a terminal state (one of "complete", "processed", "failed" or "cancelled"). In this case, after the dCDN has removed such a trigger, it MUST respond to subsequent requests for it with the HTTP error 404 ("Not Found") and remove it from all trigger collections.

If the dCDN does remove triggers in a terminal state automatically, it MUST report the expiry timeout period, using an attribute "staleresourcetime" of the trigger index resource (see Section 4.2 for details.

It is RECOMMENDED that the dCDN sets the value of the "staleresourcetime" attribute to at least 24 hours. It is further RECOMMENDED that the uCDN sets its trigger polling period to less than this period, so it doesn’t miss trigger status updates before the "complete" or "failed" triggers are expired by the dCDN.

### 3.7. Error Handling

The dCDN MAY reject CI/T interface requests by responding with 4xx or 5xx HTTP status codes. For example, the uCDN MAY respond with 400 ("Bad Request") if the request is malformed, or 403 ("Forbidden") or 404 ("Not Found") if the request could not be properly authenticated or if the uCDN is trying to act on another CDN’s resources.

If any part of the trigger processing fails, the trigger SHOULD be reported as "failed" once its activity is complete or if no further errors will be reported. The "errors" property in the trigger will be used to enumerate which actions failed and the reasons for failure, and can be present while the trigger is still "pending" or "active" if the trigger processing is still running for some URLs or patterns in the trigger specs.

Once a request has been accepted, processing errors are reported in the trigger using a list of Error.v2 Descriptions. Each Error.v2 Description is used to report errors against one or more of the URLs or patterns in the trigger specification.

If a Surrogate affected by a trigger is offline in the dCDN or the dCDN is unable to pass a trigger on to any of its cascaded dCDNs:

* If the trigger is abandoned by the dCDN, the dCDN SHOULD report an error.

* A CI/T "invalidate" command may be reported as "complete" when Surrogates that may have the data are offline. In this case, Surrogates MUST NOT use the affected data without first revalidating it when they are back online.

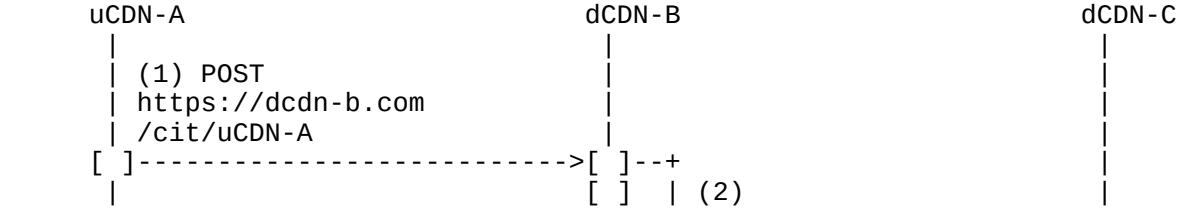
* CI/T "preposition" and "purge" commands can be reported as "processed" if affected caches are offline and the activity will complete when they return to service.

* Otherwise, the dCDN SHOULD keep the trigger "pending" or "active" state until either the trigger is acted upon or the uCDN chooses to cancel it.

### 3.7.1. Error Propagation

This subsection explains the mechanism for enabling the uCDN to trace an error back to the dCDN in which it occurred. CDNI triggers may be propagated over a chain of downstream CDNs. For example, an upstream CDN A (uCDN-A) that is delegating to a downstream CDN B (dCDN-B) and dCDN-B is delegating to a downstream CDN C (dCDN-C). Triggers sent from uCDN-A to dCDN-B may be redistributed from dCDN-B to dCDN-C, and errors can occur anywhere along the path. Therefore, it might be essential for uCDN-A which sets the trigger to be able to trace back an error to the downstream CDN where it occurred. This document adds a mechanism to propagate the PID of the dCDN where the fault occurred, back to the uCDN by adding the PID to the error.v2 description. When dCDN-B propagates a trigger further to dCDN-C, it MUST also propagate back the errors received in the trigger status resource from dCDN-C by adding them to the errors array in its status resource to be sent back to the originating uCDN-A. While propagating back the errors dCDN-B MAY also specify the dCDN-C PID, indicating to which CDN the error specifically relates. The trigger-originating upstream uCDN-A then receives an array containing the errors that occurred across all the downstream CDNs along the execution path, where each error MAY include its own CDN identifier.

Figure 2 below is an example showing the message flow used by uCDN-A to trigger activity in dCDN-B, followed by dCDN-C, as well as the discovery of the status of that activity, including the Error Propagation.



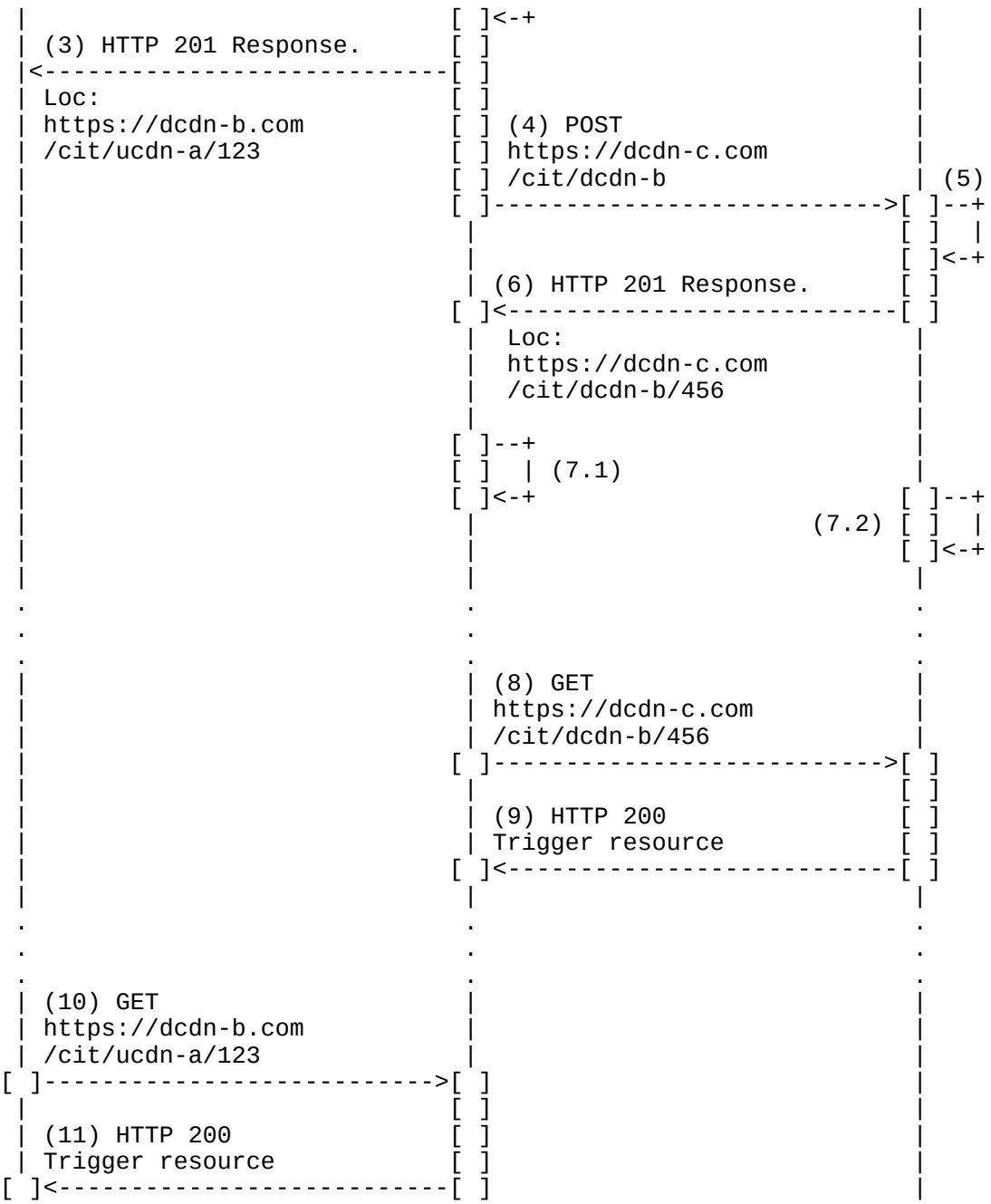



Figure 2: CDNI Message Flow for Triggers, Including Error Propagation

The steps in Figure 2 are as follows:

1. uCDN-A creates a trigger in dCDN-B by POSTing a new trigger representation to "https://dcdn-b.com/cit/ucdn-a".

2. dCDN-B authenticates the request, validates the trigger creation request, and, if it accepts the request, creates a new trigger resource.

3. dCDN-B responds to uCDN-A with an HTTP 201 ("Created") response status and the location of the newly created trigger.

4. dCDN-B creates a trigger in dCDN-C by POSTing the newly received trigger representation to "https://dcdn-c.com/cit/dcdn-b".

5. dCDN-C authenticates the request, validates the trigger creation request, and, if it accepts the request, creates a new trigger resource.

6. dCDN-C responds to dCDN-B with an HTTP 201 ("Created") response status and the location of the newly created trigger resource.

7. dCDN-C acts upon the trigger. However, the command fails at dCDN-C as, for example, the trigger definition contains an "action" type that is not supported by dCDN-C. The dCDN-C’s action is depicted by 7.2 in the diagram, while 7.1 shows dCDN-B acting on its own trigger.

8. dCDN-B queries, possibly repeatedly, the trigger resource in dCDN-C.

9. dCDN-C responds with the trigger resource representation, describing the progress or results of the trigger processing. In the described flow, the trigger state is "failed", with an Error.v2 Description object holding "eunsupported" Error Code reflecting the reason.

10. uCDN-A queries, possibly repeatedly, the trigger status in dCDN-B.

11. dCDN-B responds with the updated trigger resource representation, describing the progress or results of trigger processing. In the flow described above, the trigger state is "failed", and the "eunsupported" error received in the trigger status from dCDN-C is propagated-along with dCDN-C's PID-by adding it to the errors array in dCDN-B's status resource, which is then sent back to the originating uCDN-A. The use of dCDN-C's PID is OPTIONAL; dCDN-B MAY instead use its own PID, thereby obfuscating the identity of dCDN-C from the upstream CDN.

## 4. CI/T Object Properties and Encoding

The Trigger (Section 4.1)>, Trigger Index (Section 4.2), and Trigger Collection (Section 4.3) resources and their respective properties are encoded in JSON format. When sending the JSON-based representation of these resources, the MIME media type "application/cdni" MUST be used, with parameter "ptype" values as defined below and in Section 7.1.

Names in JSON are case-sensitive. The names and literal values specified in the present document MUST always use lowercase.

JSON types, including "object", "array", "number", and "string", are defined in [RFC8259].

Unrecognized name/value pairs in JSON objects SHOULD NOT be treated as an error by either the uCDN or the dCDN. They SHOULD be ignored during processing and passed on by the dCDN to any further dCDNs in a cascade.

### 4.1. Trigger Resource

Trigger resource is encoded as a JSON object and MUST use a MIME media type of "application/cdni; ptype=ci-trigger.v2". Please note that the dCDN MUST include all existing trigger attributes in the trigger resource representation when requested by the uCDN. The trigger resource contains the following name/value pairs:

Name: action
Description: Defines the type of the CI/T trigger action.

Value: Trigger action type, as defined in Section 4.1.1.

Mandatory-to-Specify: It is optional for trigger updates, otherwise mandatory.

Name: specs

Description: Array of trigger specs representing the trigger's targets, as described in Section 4.1.2.

Value: Array of GenericTriggerSpec objects (see Section 4.1.2.1).

Mandatory-to-Specify: It is optional for trigger updates, otherwise mandatory. Furthermore, when mandatory, the list MUST NOT be empty.

Name: extensions
Description: Array of trigger extensions, as described in Section 4.1.3.

Value: Array of GenericTriggerExtension objects (see Section 4.1.3.2).

Mandatory-to-Specify: No. The default is no extensions.

Name: labels
Description: Array of trigger labels, as described in Section 4.1.4.

Value: Array of trigger labels. Each label is a key-value pair, encoded as a JSON string, with "=" separator. The label key and value parts MUST contain up to 63 characters each, MUST begin with a letter or a number, and MAY contain letters, numbers, hyphens, dots, and underscores.

Mandatory-to-Specify: No. The default is no labels.

Name: cdn-path
Description: The chain of CDN PIDs of CDNs that have already created this trigger resource.

Value: Non-empty array of JSON strings, where each string is a CDN PID as defined in Section 4.4.3.

Mandatory-to-Specify: No. The default is no CDN path.

Name: ctime
Description: The time at which the trigger resource was received by the dCDN. The time is determined by the dCDN; there is no requirement to synchronize clocks between interconnected CDNs.

Value: Time, as defined in Section 4.3.4 of [RFC8006].

Mandatory-to-Specify: The dCDN MUST specify in trigger status representations. It is ignored when included in trigger representations sent by the uCDN.

Name: mtime
Description: The time at which the trigger resource was last modified. The time is determined by the dCDN; there is no requirement to synchronize clocks between interconnected CDNs.

Value: Time, as defined in Section 4.3.4 of [RFC8006].

Mandatory-to-Specify: The dCDN MUST specify in trigger status representations. It is ignored when included in trigger representations sent by the uCDN.

Name: etime
Description: The estimate of the time at which the dCDN expects to complete the trigger processing. Time is determined by the dCDN; there is no requirement to synchronize clocks between interconnected CDNs.

Value: Time, as defined in Section 4.3.4 of [RFC8006].

Mandatory-to-Specify: The dCDN MAY specify in trigger status representations. It is ignored when included in trigger representations sent by the uCDN.

Name: state
Description: The current trigger state.

Value: Trigger state, as defined in Section 4.1.5.

Mandatory-to-Specify: The dCDN MUST include trigger state in the trigger resource representations it sends. The trigger state defaults to "pending" when a trigger is created and is optional in trigger updates sent by the uCDN.

Name: state-reason
Description: A human-readable explanation for the object state.

Value: A JSON string, the human-readable reason.

Mandatory-to-Specify: No. The dCDN MAY include the trigger reason in the trigger resource representations it sends.

Name: errors
Description: Descriptions of errors that have occurred while processing the trigger.

Value: An array of Error.v2 Descriptions, as defined in Section 4.1.6.1. An empty array is allowed and is equivalent to omitting the "errors" attribute from the object.

Mandatory-to-Specify: No. The dCDN SHOULD include this attribute in the trigger resource representations it sends when the trigger is in a "failed" state.

Name: total-objects-count

Description: Total aggregate number of objects affected by the trigger, e.g., number of objects purged, invalidated or prepositioned as a result of trigger processing. It is RECOMMENDED that the dCDN maintains this attribute as a cumulative counter across all of its nodes, without de-duplicating the same objects processed in multiple nodes. The primary purpose of this attribute is to help the uCDN identify abnormal trigger processing results, e.g., a purge or preposition trigger that impacted a lower number of objects than expected. The dCDN MAY provide an updated object count as the trigger processing progresses in an "active" trigger state.

Value: Integer.

Mandatory-to-Specify: No. This attribute is "optional-to-implement". When supported, the dCDN SHOULD include this attribute in the trigger resource representations requested by the uCDN.

Name: total-nodes-count

Description: The total number of unique dCDN nodes affected by the trigger. The primary purpose of this attribute is to help the uCDN validate trigger processing results in combination with the cumulative counters "total-objects-count" and "total-objects-size". The dCDN MAY provide an updated nodes count as the trigger processing progresses in an "active" trigger state.

Value: Integer.

Mandatory-to-Specify: No. This attribute is "optional-to-implement". When supported, the dCDN SHOULD include this attribute in the trigger resource representations requested by the uCDN.

Name: total-objects-size

Description: Total aggregate size of objects affected by the trigger, in bytes. It is RECOMMENDED that the dCDN maintains this attribute as a cumulative counter across all of its nodes, without de-duplicating the same objects processed in multiple

nodes. As with "total-objects-count" attribute above, the primary purpose of this attribute is to help the uCDN identify abnormal trigger processing results. The dCDN MAY provide an updated object size as the trigger processing progresses in "active" trigger state.

Value: Integer.

Mandatory-to-Specify: No. This attribute is "optional-to-implement". When supported, the dCDN SHOULD include this attribute in the trigger resource representations requested by the uCDN.

Name: objects

Description: List of objects derived by the dCDN when processing the trigger. When the "content-objectlist" trigger specification is used, the trigger MAY include one or more manifest files that form a hierarchical structure (e.g., media HLS playlists referenced by master HLS playlists, or JSON object lists referenced by other JSON object lists). The purpose of this optional attribute is to provide a single, flattened list of content objects derived from the input trigger.

Value: An array of ObjectList (Section 4.4.2) objects. The dCDN SHOULD provide the list of objects it used as input for processing the trigger with Section 4.1.2.8, provided that the dCDN advertised support for extended status (Section 5.5). An empty array is allowed and is equivalent to omitting "objects" from the trigger representation. This field is intended to provide the list of all objects used in processing. The objects that failed to process SHOULD be specified using the Error.v2 Description resource.

Mandatory-to-Specify: No. The dCDN MAY send this attribute in its trigger resource representation when available and the capability is advertised via FCI as described above.

#### 4.1.1. Trigger Action

A trigger action is used in a trigger resource to describe trigger actions. It was initially referred to in [RFC8007] as "Trigger Type".

All trigger actions MUST be registered in the IANA "CDNI CI/T Trigger Types" registry (see Section 7.2).

The dCDN receiving a request containing a trigger action that it does not recognize or does not support MUST reject the request by creating a trigger with a "failed" state and the "errors" array containing an Error.v2 Description with error "eunsupported" (see Section 4.1.6.2).

The following trigger actions are defined by this document:

| JSON string | Description |
|---|---|
| preposition | A request for the dCDN to acquire metadata or content. |
| invalidate | A request for the dCDN to invalidate metadata or content. After servicing this request, the dCDN will not use the specified data without first revalidating it using, for example, an "If-None-Match" HTTP request. The dCDN need not erase the associated data. |
| purge | A request for the dCDN to erase metadata or content. After servicing the request, the specified data MUST NOT be held on the dCDN (the dCDN should reacquire the metadata or content from the uCDN if it needs it). |

Table 2

The dCDN MUST support at least one of the trigger actions.

### 4.1.2. Trigger Specs

The CDNI Control Interface / Triggers 1st edition [RFC8007] defines a set of properties and objects used by the trigger commands to specify the targets upon which the trigger is applied. This document modifies the trigger interface objects so that it has a list of trigger specs. Such structure improves the interface's extensibility and flexibility. Furthermore, the document defines a generic trigger spec object that acts as a wrapper for managing individual CDNI trigger specs in an opaque manner, allowing future extension of the interface.

All trigger specs MUST be registered in the IANA "CDNI CI/T Trigger Specs" registry (see Section 7.3).

A dCDN receiving a trigger creation request that contains a trigger spec it does not recognize or support MUST reject the request by creating a trigger resource with a "failed" state and the "errors" array containing an Error.v2 Description with the error "espec" (see Section 4.1.6.2).

This document also defines an initial set of trigger spec objects and registers them in the IANA "CDNI CI/T Trigger Specs" registry:

| JSON string | Description |
|---|---|
| urls | Allowing the specification of trigger targets via URLs. |
| ccids | Allowing the specification of trigger targets via CCIDs content grouping, as defined in section 4.2.8 [RFC8006]. |
| uri-pattern-match | Allowing the specification of trigger targets via [RFC3986] URI patterns. |
| uri-regex-match | Allowing the specification of trigger targets via regexes matching their URI, as defined in Section 4.1.2.7. |
| content-objectlist | Allowing the specification of trigger targets via an object list and an object list type. |

Table 3

The dCDN MUST support the "urls" trigger spec. Support for all other trigger specs is OPTIONAL.

Each trigger usually refers to the targets by the target URLs, using a "urls" trigger spec object or some aggregating spec such as the "url-regex-match". If content URLs are transformed by a transit CDN in a cascade, that transit CDN MUST similarly transform URLs in triggers it passes to its dCDNs.

When processing a trigger, CDNs MUST ignore the URL scheme (HTTP or HTTPS) in comparing URLs. For example, for a CI/T "invalidate" or "purge" action, content MUST be invalidated or purged regardless of the protocol clients used to request it.

#### 4.1.2.1. Generic Spec Object

A trigger resource, as defined in Section 4.1, includes an array of trigger spec objects. Each trigger spec object contains properties that are used as trigger target selection directives for the dCDN when processing the trigger, e.g., content URLs or metadata URL patterns. Each such trigger spec is a specialization of a CDNI GenericTriggerSpec object. The GenericTriggerSpec object abstracts the basic information required for trigger distribution from the specifics of any given property (i.e., property semantics, enforcement options, etc.).

The semantics of the trigger specs list is additive, i.e., the trigger applies to any object matching one of the listed specs.

A GenericSpecObject object is a wrapper for managing individual CDNI trigger specs in an opaque manner.

It is encoded as a JSON object containing the following name/value pairs:

Name: trigger-subject
Description: Case-insensitive CDNI trigger subject.

Value: String containing the type of the subject matching the "cit-spec-value" property, such as "content" or "metadata" as defined in Section 4.1.2.2.

Mandatory-to-Specify: Yes.

Name: cit-spec-type
Description: Case-insensitive CDNI trigger spec type.

Value: String containing the spec type of the object contained in the cit-spec-value property (see table in Section 4.1.2).

Mandatory-to-Specify: Yes.

Name: cit-spec-value
Description: A CDNI trigger spec object.

Value: Defined by the value of the cit-spec-type property.

Mandatory-to-Specify: Yes.

The structure of a JSON-serialized GenericTriggerSpec object, containing a specific trigger spec is illustrated below:

```
{
  "cit-spec-type":
     <Type of this trigger spec>,
  "cit-spec-value":
      {
        <properties of this trigger spec object>
      },
   "trigger-subject":
     <Category of this trigger spec subject>

}
```

#### 4.1.2.2. Trigger Subject

Because the scope of the trigger may relate to either metadata as well as content, the "trigger spec object" also specifies the trigger's target subject (i.e., metadata or content) against which to match.

All trigger subjects MUST be registered in the IANA "CDNI CI/T Trigger Subjects" registry (see Section 7.4).

The dCDN receiving a trigger creation request containing a trigger subject that it does not recognize or does not support MUST reject the request by creating a trigger resource with a "failed" state and the "errors" array containing an Error.v2 Description with error "esubject" (see Section 4.1.6.2).

This document also defines an initial set of trigger subject values and registers them in the IANA "CDNI CI/T Trigger Subjects" registry:

| JSON string | Description |
|---|---|
| metadata | Indicating the trigger target specification refers to Metadata object(s), as defined at [RFC8006]. |
| content | Indicating the trigger target specification refers to client-facing content objects. |

Table 4

The dCDN MUST support at least one of the trigger subjects. The dCDN MAY advertise separate endpoints for each one of the two trigger subjects, using the CI/T Endpoint Capability Object (Section 5.1).

#### 4.1.2.3. Spec Constraints

There are certain constraints on the way the trigger specs can be combined with trigger subject and trigger actions:

| Trigger spec types | Trigger subject ("content" and "metadata") | Trigger action ("preposition", "purge", and "invalidate") |
|---|---|---|
| urls | Any | Any |
| ccids | "content" only | "purge" or "invalidate" |
| uri-pattern-match | Any | "purge" or "invalidate" |
| uri-regex-match | Any | "purge" or "invalidate" |
| content-objectlist | Any | Any |

Table 5: Summary of trigger spec constraints

#### 4.1.2.4. URLs Spec

The "urls" spec type allows the uCDN to manage uCDN content or metadata objects held by the dCDN based on the objects' URLs. Full URLs SHOULD be used to ensure unambiguous identification of the referenced objects.

The URLs spec is encoded as a JSON object containing the following name/value pairs:

Name: urls
Description: An array of URLs over which the trigger MUST be executed.

Value: A JSON array of URLs, each represented as a JSON string.

Mandatory-to-Specify: Yes.

Name: url-type
Description: Type of URL used.

Value: URL Type as defined in Section 4.4.1.

Mandatory-to-Specify: No. When omitted or empty, the "published" URL type is assumed.

Below is an example of a JSON-serialized URLs spec object, matching the metadata at metadata.example.com/a/b/c.

```
{
  "trigger-subject": "metadata",
  "cit-spec-type": "urls",
  "cit-spec-value": {
    "urls": [ "https://metadata.example.com/a/b/c" ],
    "url-type": "published"
  }
}
```

#### 4.1.2.5. CCIDs Spec

The "ccids" spec type allows the uCDN to specify the Content Collection IDentifier (CCID) of content to which the trigger applies. The CCID is a grouping of content as defined by [RFC8006]. The "ccids" spec type is valid only for the "content" spec subject (see Section 4.1.2.2).

CCIDs spec is encoded as a JSON object containing the following name/value pairs:

Name: ccids
: Description: An array of Content Collection IDentifiers over which the trigger MUST be executed.

  Value: A JSON array of strings, where each string is a Content Collection IDentifier.

  Mandatory-to-Specify: Yes.

#### 4.1.2.6. URI Pattern Match Spec

The "uri-pattern-match" spec type allows the uCDN to manage its content or metadata objects held by the dCDN based on the objects' URI pattern. The value is a UriPatternMatch object, as defined in Section 4.1.2.6.1.

##### 4.1.2.6.1. UriPatternMatch

A UriPatternMatch consists of a string pattern to match against a URI, and flags describing the type of match.

It is encoded as a JSON object containing the following name/value pairs:

Name: pattern

Description: A pattern for URI matching.

Value: The pattern represented as a JSON string. The pattern can contain the wildcards "*" and "?", where "*" matches any sequence of [RFC3986] pchar or "/" characters (including the empty string) and "?" matches exactly one [RFC3986] pchar character. The three literals "$", "*", and "?" MUST be escaped as "$$", "$*" and "$?" (where "$" is the designated escape character). All other characters are treated as literals.

Mandatory-to-Specify: Yes.

Name: case-sensitive
Description: Flag indicating whether or not case-sensitive matching should be used.

Value: A JSON boolean. When set to "true", matching is case-sensitive; when set to "false", matching is case-insensitive.

Mandatory-to-Specify: No; default is "false", i.e., a case-insensitive match.

Name: match-query-string
Description: Flag indicating whether to include the query part of the URI when compared against the pattern.

Value: One of the JSON values "true" (the full URI including the query part should be compared against the given pattern) or "false" (the query part of the URI should be dropped before comparison with the given pattern).

Mandatory-to-Specify: No; default is "false". The query part of the URI should be dropped before comparison with the given pattern.

Name: url-type
Description: Type of URLs to match.

Value: URL Type as defined in Section 4.4.1.

Mandatory-to-Specify: No. When omitted or empty, "published" URL type is assumed.

Example of case-sensitive prefix match against "https://www.example.com/trailers/":

```
{
  "pattern": "https://www.example.com/trailers/*",
  "case-sensitive": true
}
```

#### 4.1.2.7. URI Regex Match Spec

The "uri-regex-match" spec type allows the uCDN to manage content or metadata objects held by the dCDN based on the objects' URI regex.

##### 4.1.2.7.1. RegexMatch

A RegexMatch consists of a regular expression string a URI is matched against, and flags describing the type of match. It is encoded as a JSON object with the following properties:

Name: regex
Description: A regular expression for URI matching.

Value: A regular expression to match against the URI, i.e., against the path-absolute and the query string parameters [RFC3986]. The regular expression string MUST be compatible with POSIX [POSIX.1] Section 9 Extended Regular Expressions. This regular expression MUST be evaluated in the POSIX locale (POSIX [POSIX.1] Section 7.2).

Note: Because '\' has a special meaning in JSON [RFC8259] as the escape character within JSON strings, the regular expression character '\' MUST be escaped as '\\'.

Mandatory-to-Specify: Yes.

Name: case-sensitive
Description: Flag indicating whether or not case-sensitive matching should be used.

Value: JSON boolean. Either "true" (the matching is case-sensitive) or "false" (the matching is case insensitive).

Mandatory-to-Specify: No; default is "false", i.e., a case-insensitive match.

Name: match-query-string
Description: Flag indicating whether to include the query part of the URI when compared against the regex.

Value: JSON Boolean. Either "true" (the full URI, including the query part, should be compared against the regex) or "false" (the query part of the URI should be dropped before comparison with the given regex).

Mandatory-to-Specify: No; default is "false". The query part of the URI MUST be dropped before comparison with the given regex. This makes the regular expression simpler and safer for cases in which the query parameters are not relevant to the match.

Name: url-type
Description: Type of URLs to match against.

Value: URL Type as defined in Section 4.4.1.

Mandatory-to-Specify: No. When omitted or empty, "published" URL type is assumed.

Example of a case-sensitive, no query parameters, regex match against is below.

Please note that some lines in the example are wrapped for clarity.

```
"^(https:\/\/video\.example\.com)\/([a-z])\/
 movie1\/([1-7])\/*(index.m3u8|\d{3}.ts)$"
```

```
{
  "regex": "^(https:\\/\\/video\\.example\\.com)\\/([a-z])\\/
           movie1\\/([1-7])\\/*(index.m3u8|\\d{3}.ts)$",
  "case-sensitive": true
}
```

This regex matches URLs of the domain "video.example.com" where the path structure is /(single lower case letter)/(name-of-title)/(single digit between 1 to 7)/(index.m3u8 or a 3 digit number with ts extension). For example:

https://video.example.com/d/movie1/5/index.m3u8

or

https://video.example.com/k/movie1/4/013.ts

#### 4.1.2.8. Object List Spec

The "objectlist" spec type allows the uCDN to manage content or metadata held by the dCDN based on structured object lists. The Object List spec type is valid only for the "content" spec subject (see Section 4.1.2.2).

An object list is encoded as a JSON object with the following properties:

Name: objects
Description: An array of objects to be used in the trigger

Value: Array of ObjectList (Section 4.4.2) objects

Mandatory-to-Specify: Yes.

### 4.1.3. Trigger Extensions

A "trigger" object, as defined in Section 4.1 includes an optional array of trigger extension objects. A trigger extension contains properties that are used as directives for the dCDN when executing the trigger command, e.g., location policies, time policies, and so on. Each such CDNI trigger extension is a specialization of a CDNI GenericTriggerExtension object. The GenericTriggerExtension object abstracts the basic information required for trigger distribution from the specifics of any given property (i.e., property semantics, enforcement options, etc.). All trigger extensions are optional, and it is thus the responsibility of the extension specification to define a consistent default behavior for extensions supported by the dCDN when not specified by the uCDN.

#### 4.1.3.1. Enforcement Options

The trigger enforcement options concept is in accordance with the metadata enforcement options as defined in Section 3.2 of [RFC8006].

The GenericTriggerExtension object defines the properties contained within it as well as whether or not the properties are "mandatory-to-enforce". If the dCDN does not understand or support a mandatory-to-enforce property, the dCDN MUST NOT execute the trigger command. If the extension is not mandatory-to-enforce, then that GenericTriggerExtension object can be safely ignored and the trigger command can be processed in accordance with the rest of the CDNI trigger spec.

Although a CDN MUST NOT execute a trigger command if a mandatory-to-enforce extension cannot be enforced, it could still be safe for a transit CDN (tCDN) to redistribute that trigger (the "safe-to-redistribute" property) to another CDN without modification, provided the tCDN does not need to do trigger processing of its own and only pass the trigger to one or more dCDNs. For example, in the cascaded CDN case, a transit CDN (tCDN) could convey mandatory-to-enforce trigger extension to the dCDN. For a trigger extension that does not require customization or translation (i.e., trigger extension that is safe-to-redistribute), the data representation received off the wire MAY be stored and redistributed without being understood or supported by tCDN. However, for a trigger extension that requires translation, transparent redistribution of the uCDN trigger values might not be appropriate. Certain trigger extensions can be safely, though perhaps not optimally, redistributed unmodified. For example, a preposition command might be executed in suboptimal times for some geographies if transparently redistributed, but it might still work.

Redistribution safety MUST be specified for each GenericTriggerExtension listed. If a CDN does not understand or support a given GenericTriggerExtension object that is not safe-to-redistribute, the CDN MUST set the "incomprehensible" flag to true for that GenericTriggerExtension object before redistributing it. The "incomprehensible" flag signals to the dCDN that trigger metadata was not properly transformed by the tCDN. A CDN MUST NOT attempt to execute a trigger with an extension that has been marked as "incomprehensible" by the uCDN.

tCDNs MUST NOT change the value of mandatory-to-enforce or safe-to-redistribute when propagating a trigger to the dCDN. Although a tCDN can set the value of "incomprehensible" to true, a tCDN MUST NOT change the value of "incomprehensible" from true to false.

Table 6 describes the action to be taken by a tCDN for the different combinations of mandatory-to-enforce ("MtE") and safe-to-redistribute ("StR") properties when the tCDN either does or does not understand the trigger extension object in question:

| MtE | StR | Extension object understood by tCDN | Trigger action |
|---|---|---|---|
| False | True | True | Can execute and redistribute. |
| False | True | False | Can execute and redistribute. |
| False | False | False | Can execute. MUST set "incomprehensible" to true when redistributing. |
| False | False | True | Can execute. Can redistribute after transforming the trigger extension (if the CDN knows how to do so safely); otherwise, MUST set "incomprehensible" to true when redistributing. |
| True | True | True | Can execute and redistribute. |
| True | True | False | MUST NOT execute but can redistribute, provided own processing is not required. |
| True | False | True | Can execute. Can redistribute after transforming the trigger extension (if the CDN knows how to do so safely); otherwise, MUST set "incomprehensible" to true when redistributing. |
| True | False | False | MUST NOT execute. May redistribute, provided own processing is not required. MUST set "incomprehensible" to true when redistributing. |

Table 6: Action to be taken by a tCDN for the different combinations of MtE and StR properties

Table 7 describes the action to be taken by the dCDN for the different combinations of mandatory-to-enforce and "incomprehensible" ("Incomp") properties, when the dCDN either does or does not understand the trigger extension object in question:

| MtE | Incomp | Extension object understood by the dCDN | Trigger action |
|---|---|---|---|
| False | False | True | Can execute. |
| False | True | True | Can execute but MUST NOT interpret/apply any trigger extension marked as "incomprehensible". |
| False | False | False | Can execute. |
| False | True | False | Can execute but MUST NOT interpret/apply any trigger extension marked as "incomprehensible". |
| True | False | True | Can execute. |
| True | True | True | MUST NOT execute. |
| True | False | False | MUST NOT execute. |
| True | True | False | MUST NOT execute. |

Table 7: Action to be taken by the dCDN for the different combinations of MtE and Incomp properties

#### 4.1.3.2. GenericExtensionObject

A GenericTriggerExtension object is a wrapper for managing individual CDNI Trigger extensions in an opaque manner.

It is encoded as a JSON object containing the following name/value pairs:

Name: cit-extension-type
Description: Case-insensitive CDNI trigger extension object type.

Value: String containing the CDNI Extension Type [RFC7736] of the object contained in the "cit-extension-value" property (see table in Section 2.8).

Mandatory-to-Specify: Yes.

Name: cit-extension-value
Description: CDNI trigger extension object.

Value: Defined by the value of the "cit-extension-type" property above.

Mandatory-to-Specify: Yes.

Name: mandatory-to-enforce
Description: Flag identifying whether or not the enforcement of this trigger extension is mandatory.

Value: Boolean.

Mandatory-to-Specify: No. The default is to treat the trigger extension as mandatory to enforce (i.e., a value of True).

Name: safe-to-redistribute
Description: Flag identifying whether or not this trigger extension can be safely redistributed without modification, even if the CDN fails to understand the extension.

Value: Boolean.

Mandatory-to-Specify: No. The default is to allow transparent redistribution (i.e., a value of True).

Name: incomprehensible
Description: Flag identifying whether or not any CDN in the chain of delegation has failed to understand and/or failed to properly transform this trigger extension object. Note: This flag only applies to trigger extension objects whose "safe-to-redistribute" property has a value of False.

Value: Boolean.

Mandatory-to-Specify: No. The default is comprehensible (i.e., a value of False).

The structure of a JSON-serialized GenericTriggerExtension object containing a specific trigger extension object is illustrated below:

```
{
  "cit-extension-type":
     <Type of this trigger extension object>,
  "cit-extension-value":
      {
        <properties of this trigger extension object>
      },
  "mandatory-to-enforce": <bool>,
  "safe-to-redistribute": <bool>,
  "incomprehensible": <bool>
}
```

#### 4.1.3.3. Trigger Extension Objects

The objects defined below are intended to be used in the GenericTriggerExtension object's cit-extension-value field as defined in Section 4.1.3.2, and their cit-extension-type property MUST be set to the appropriate Extension Type as defined in Section 2.8.

##### 4.1.3.3.1. LocationPolicy Extension

A content operation may be relevant for a specific geographical region or need to be excluded from a specific region. In this case, the trigger should be applied only to parts of the network that are either "included" or "not excluded" by the location policy. Note that the restrictions here are on the cache location rather than the client location.

The LocationPolicy object defines the cache or CDN locations in which the trigger is to be executed, thereby constraining the trigger's scope to those locations. Although users are typically proximate to the corresponding cache nodes, the policy applies to the caches themselves, not to the users' locations.

Example use cases:

* Preposition: Because modern streaming content often includes numerous renditions - across resolutions, bitrates, and languages - prepositioning all variants can be untenable. The uCDN MAY perform location-aware prepositioning by selecting which content renditions to preposition based on the languages prevalent in a given region. For example, only Danish, Norwegian and Swedish audio or subtitle renditions might be prepositioned in southern Scandinavia.

* Purge: In certain cases, content may have been located on servers in regions where the content must not reside. In such cases, a purge operation to remove content specifically from that region is required.

Object specification:

Name: locations
Description: An Access List that allows or denies (blocks) the trigger execution per cache location.

Value: Array of LocationRule objects (see Section 4.2.2.1 of [RFC8006]). The LocationRule utilizes Footprint objects to define footprints in which the rule is to be applied, as defined in Section 4.2.2.2 of [RFC8006] and extended by [RFC9388] to support the "subdivisioncode" footprint type.

Mandatory-to-Specify: Yes.

If a location policy object is not listed within the trigger command, the default behavior is to execute the trigger in all available caches and locations of the dCDN.

The trigger command is allowed or denied for a specific cache location according to the action of the first location whose footprint matches that cache’s location. The evaluation order is implicit and follows the sequence of LocationRule objects as they appear in the extension. If two or more footprints overlap, the first footprint that matches against the cache’s location determines the action a CDN MUST take.

If the "locations" property is an empty list, or if none of the listed footprints match the location of a given cache, the trigger MUST be treated as not applicable to that cache - that is, the result is equivalent to a "deny" action.

The following is an example of a JSON-serialized generic trigger extension object containing a location policy object that allows the trigger execution in the US but blocks its execution in Massachussetts. The execution of the trigger outside of the US is blocked implicitly.

```
{
  "cit-extension-type": "location-policy",
  "cit-extension-value": {
    "locations": [
      {
        "action": "allow",
        "footprints": [{
          "footprint-type": "countrycode",
          "footprint-value": [ "us" ]
        }]
      },
      {
        "action": "deny",
        "footprints": [{
          "footprint-type": "subdivisioncode",
          "footprint-value": [ "us-ma" ]
        }]
      }
    ]
  }
}
```

#### 4.1.3.3.2. TimePolicy Extension

The uCDN may wish to perform content management operations on the dCDN on a specific schedule. The TimePolicy extension allows the uCDN to instruct the dCDN to execute the trigger command in a desired time window. For example, a video content provider may wish to pre-populate a new episode during off-peak hours so that it is ready on caches at prime time when the episode is released for viewing. A scheduled operation enables the uCDN to direct the dCDN in what time frame to execute the trigger.

This specification supports region-by-region time scheduling when used in conjunction with the Location Policy defined in Section 4.1.3.3.1. The uCDN can trigger separate commands for different geographical regions using a different schedule for each region. This allows the uCDN to control the execution time per region.

Object specification:

Name: unix-time-window
Description: A UNIX epoch time window in which the trigger SHOULD be executed.

Value: TimeWindow object using UNIX epoch timestamps (see Section 4.2.3.2 of [RFC8006]).

Mandatory-to-Specify: No, but exactly one of either "unixEpochWindow" or "utcWindow" MUST be present.

Name: utc-window
Description: A UTC time window in which the trigger SHOULD be executed.

Value: UTCWindow object as defined in Section 4.1.3.3.2.1.

Mandatory-to-Specify: No, but exactly one of either "unixEpochWindow" or "utcWindow" MUST be present.

If a time policy object is not listed within the trigger command, the default behavior is to execute the trigger in a time frame most suitable to the dCDN taking under consideration other constraints and / or obligations.

Example of a JSON-serialized generic trigger extension object containing a time policy object that schedules the trigger execution to a window between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC, using the "unix-time-window" property:

```
{
   "cit-extension-type": "time-policy",
   "cit-extension-value":
    {
      "unix-time-window": {
         "start": 946717200,
         "end": 946746000
      }
    },
   "mandatory-to-enforce": true,
   "safe-to-redistribute": true,
   "incomprehensible": false
}
```

#### 4.1.3.3.2.1. UTCWindow

A UTCWindow object describes a time range in UTC or UTC and a zone offset that can be applied by a TimePolicy.

It is encoded as a JSON object containing the following name/value pairs:

Name: start
Description: The start time of the window.

Value: Internet date and time as defined in [RFC3339].

Mandatory-to-Specify: No. but at least one of "start" or "end" MUST be present and non-empty. If "start" is empty or not specified, the time range is considered to begin at an arbitrary (unspecified) time.

Name: end
Description: The end time of the window.

Value: Internet date and time as defined in [RFC3339].

Mandatory-to-Specify: No. but at least one of "start" or "end" MUST be present and non-empty. If "end" is empty or not specified, the time range is considered to end at an arbitrary (unspecified) time.

Example JSON-serialized UTCWindow object that describes a time window from 02:30 01/01/2000 UTC to 04:30 01/01/2000 UTC:

```
{
  "start": "2000-01-01T02:30:00.00Z",
  "end": "2000-01-01T04:30:00.00Z"
}
```

Example JSON-serialized UTCWindow object that describes a time window in New York time zone offset UTC-05:00 from 02:30 01/01/2000 to 04:30 01/01/2000:

```
{
  "start": "2000-01-01T02:30:00.00-05:00",
  "end": "2000-01-01T04:30:00.00-05:00"
}
```

#### 4.1.3.3.3. ExecutionPolicy Extension

Unless specified otherwise, the dCDN is at liberty to decide how to choose trigger commands for execution from all pending commands, whether to process trigger commands sequentially or in parallel, immediately upon acceptance, or with a delay in batches. The uCDN may wish to control trigger processing in more detail, including the order of execution, dependencies, and concurrency.

Please note that the uCDN MAY request immediate processing of a trigger either by setting its state to "active" at creation time or by modifying a pending trigger to set its state to "active".

Example use cases:

* Priority: The uCDN may have multiple trigger commands in "pending" and/or "active" mode. For example, trigger commands with policy constraints, a large number of content objects affected, or other dCDN business logic may take a long time to execute. The uCDN may wish to prescribe the order in which the dCDN picks up its trigger commands for execution from the "pending" queue, by indicating a relative priority of each trigger. The priority would affect the selection of trigger commands specific to the requesting uCDN. The dCDN may separately prioritize triggers from multiple uCDNs subject to its business logic. Additionally, the uCDN may wish to prescribe the order in which parts of "active" triggers are processed. The priority may affect the order within the same trigger and/or multiple triggers that are in "active" state at the same time. Multiple priority-related use cases exist:

  - The uCDN needs to introduce an urgent "purge" or "invalidate" trigger into an existing queue of trigger commands to correct wrong versions of content objects published by it

  - The uCDN needs to indicate which content objects should be prepositioned, purged, or invalidated first, for example prepositioning newer released content before prepositioning updates to an existing catalog

* Prerequisite: In some cases, the uCDN may wish to indicate what trigger commands should be processed and completed before another trigger command is processed. For example, the uCDN may want to rectify incorrectly published content by purging content objects and then prepositioning them again. In this case, the uCDN may want the preposition trigger command to be processed only after the purge trigger command has been processed because the concurrent processing of these triggers may cause the new version of these content objects to be purged. Alternatively, the uCDN may wish to condition the execution of purge or invalidation triggers upon the completion or cancellation of long-running preposition triggers to avoid race conditions that would result from processing these in parallel. The prerequisite requirement implies that a previous trigger reaches one of the following states:

  - "complete" or "processed" for successful completion

  - "failed" for failed processing

  - "cancelled" for completion of cancellation

The ExtensionPolicy extension is encoded as a JSON object containing the following name/value pairs:

Name: priority
Description: Relative weight of the trigger. When picking a trigger for execution from all pending triggers posted by each uCDN, the dCDN MUST choose the trigger with the highest priority first.

Value: Integer from -100 to 100.

Mandatory-to-Specify: No. The value defaults to zero if omitted.

Name: depends
Description: Links to trigger resources that the current trigger depends on. Indicates which triggers should fully finish processing before starting execution of the current trigger. The triggers SHOULD be in one of the following states to be considered finished: "complete", "processed", "failed" or "cancelled".

Value: A JSON array of zero or more URLs, represented as JSON strings.

Mandatory-to-Specify: No. In case of a missing or an empty list, no dependencies are assumed.

If the dCDN receives a new trigger with the ExecutionPolicy extension that does not reference an existing trigger URL in the "depends" extension attribute, the dCDN MUST set the trigger state to "failed", set the error to "eextension", and MAY include an optional error description. A trigger modification request that would result in the "depends" attribute containing invalid trigger URLs MUST be rejected by the dCDN with status code 409 (Conflict). URL values that reference existing finished triggers are allowed but ignored.

The following is an example of a JSON-serialized generic extension trigger object containing an execution policy object that specifies trigger priority of 100, and makes its execution dependent on the completion of the previously created triggers:

```
{
  "cit-extension-type": "execution-policy",
  "cit-extension-value":
  {
    "priority": 100,
    "depends": [
     "https://dcdn.example/cit/b1467469-3cf3-4613-8629-814cd938f30b",
     "https://dcdn.example/cit/c73a9911-298b-4ee3-bbab-03bce07b7d5c"
    ]
  }
}
```

#### 4.1.3.3.4. Combining Trigger Extensions

The uCDN MAY combine multiple options in the same trigger command. The dCDN determines how to handle an incoming trigger according to the following rules:

1. When the dCDN receives a request to create a trigger, it MUST reject the request if the trigger depends on other pending triggers with lower priority in order to prevent deadlocks.

2. Otherwise, if the request for new trigger resource creation sets its state to "active":

   * If the trigger has a TimePolicy extension that has a "start" attribute set in the future, the dCDN MUST reject the request.

   * Otherwise, if the trigger has an ExecutionPolicy extension, with prerequisite triggers that are not finished at the time of the request, the dCDN MUST reject the request.

   * Otherwise, if the trigger has a lower priority than pre-existing triggers in "pending" state, the dCDN MUST reject the request. Please note that the trigger has an implicit priority of zero if no priority is explicitly specified.

   * Otherwise, if the trigger has a TimeWindow that ends before the start of the TimeWindow of any trigger it depends on, or before the start of the TimeWindow of any pending trigger with higher priority, the dCDN MUST reject the request.

   * Otherwise, the dCDN MAY still reject the trigger due to its business logic (e.g,, if the dCDN processes triggers at specific times during the day).

* Otherwise, the dCDN SHOULD create the trigger in "active" state and begin its processing.

* In all of the above cases, the dCDN rejects the request by creating a trigger in a "failed" state, setting the error to "ereject" and optionally providing an error description.

3. Otherwise, the dCDN SHOULD create the new trigger resource as requested and set its state to "pending".

4. The dCDN MAY start the trigger processing at any time after the trigger creation, as long as the prerequisites are met, i.e. the start of processing is within the TimePolicy window, there are no other pending triggers with higher priority, and there are no incomplete prerequisite triggers.

5. The dCDN SHOULD periodically re-evaluate the pending trigger queue for triggers that have a TimePolicy set, to ensure that processing of such triggers completes before the corresponding TimePolicy window expires. If during such evaluation a pending trigger is deemed to have expired, the dCDN MUST set the trigger state to "failed" and the error to "ereject", optionally providing the error description.

6. Whenever a trigger reaches a terminal state (i.e., "complete", "processed", "failed", or "cancelled"), the dCDN SHOULD re-evaluate the queue of pending triggers. If during such evaluation a pending trigger is deemed to have expired, the dCDN MUST set the trigger state to "failed" and the error to "ereject", optionally providing the error description. Otherwise, the dCDN MAY start processing triggers that were previously dependent on the completed trigger.

7. Whenever a pending trigger becomes eligible for processing, the dCDN SHOULD re-evaluate the pending trigger queue. If during such evaluation a pending trigger is deemed to have expired, the dCDN MUST set the trigger state to "failed" and the error to "ereject", optionally providing the error description. Otherwise, the dCDN MAY start processing additional triggers that could not be processed earlier due to their lower priority relative to the active trigger.

8. When the dCDN receives a request to modify the priority and/or dependencies and/or TimeWindow of a pending trigger, it MUST reject the request with status 409 (Conflict) if the change would result in that trigger depending on other triggers with lower priority, or its TimeWindow ending before the start of the TimeWindow of any trigger it depends on, or before the start of the TimeWindow of any trigger with higher priority.

9. Otherwise, the dCDN MUST reject the request with status code 409 (Conflict) if the change would result in the trigger depending on other triggers that, directly or indirectly, depend on the modified trigger.

10. When the dCDN receives a request to modify the TimeWindow extension, it MUST reject the request with status code 409 (Conflict) if the modification would set the "end" time to a value in the past.

11. Otherwise, the dCDN SHOULD accept the request and evaluate whether the modified trigger becomes eligible for processing as a result of the change. If at this time, the modified trigger has a TimePolicy "end" time set in the past, the dCDN MUST set the trigger state to "failed" and the error to "ereject", optionally providing the error description.

12. When the dCDN receives a request to change the state of a pending trigger to "active", it MUST reject the request with status code 409 (Conflict) if, as a result of the change, the trigger would depend on incomplete triggers, have lower priority than other pending triggers, have a TimeWindow start time in the future, or set a new TimeWindow end time in the past.

13. Otherwise, the dCDN MAY still reject the request with status code 409 (Conflict) if the processing of the trigger could not be started immediately due to the dCDN business logic.

14. Otherwise, the dCDN MUST set the trigger state to "active" and start its processing.

#### 4.1.4. Trigger Labels

Trigger labels provide a framework for the uCDN to associate an array of key-value pairs with trigger resources.

The labels may be used to simplify the management of a large number of triggers by grouping related triggers and tracking their status using the trigger collection resource associated with the label value (see Section 4.3 for more details). In this case, the label values remain fully opaque to the dCDN and serve for trigger grouping purposes only.

Alternatively, trigger labels may be used by the uCDN to pass to the dCDN information to be used for the trigger processing. For example, in the case of the uCDN and the dCDN utilizing non-standard configuration metadata objects, the uCDN may use trigger labels to help the dCDN identify the appropriate configuration section where the trigger should be applied. The uCDN may also use the labels to pass to the dCDN freeform content metadata to inform the dCDN cache management operations.

#### 4.1.5. Trigger State

The trigger state describes the current state of the triggered activity. It MUST be one of the JSON strings in the following table:

| JSON string | Description |
|---|---|
| pending | The trigger has not yet been acted upon. |
| active | The trigger is currently being acted upon. |
| complete | The trigger processing completed successfully. |
| processed | The trigger has been created, and no further status update will be made (can be used in cases where completion cannot be confirmed). |
| failed | The trigger processing could not be completed. |
| cancelling | The trigger processing is still in progress, but the trigger has been cancelled by the uCDN. |
| cancelled | The trigger was cancelled by the uCDN. |

Table 8

Along with the trigger state, the trigger resource has a state reason property, allowing the dCDN to provide additional information for the trigger state. For example, the dCDN may indicate that the trigger state is "pending" due to one of the execution prerequisites not being fulfilled. Such a prerequisite may be specified via one of the extensions.

#### 4.1.6. Trigger Errors

##### 4.1.6.1. Error.v2 Description

An Error.v2 Description is used to report the failure of a trigger. It is encoded as a JSON object with the following name/value pairs:

Name: error
Value: Error Code, as defined in Section 4.1.6.2.

Mandatory-to-Specify: Yes.

Name: description
Description: A human-readable description of the error.

Value: A JSON string, the human-readable description.

Mandatory-to-Specify: No.

Name: specs
Description: Array of trigger spec objects from the corresponding "specs" array in the trigger resource. Only those specs to which the error applies are listed.

Value: Array of trigger specs, as defined in Section 4.1.2, where each spec object MUST be exactly as it appears in the trigger resource.

Mandatory-to-Specify: Yes.

Name: extensions
Description: Array of trigger extension objects copied from the corresponding "extensions" array in the trigger resource. Only those extensions to which the error applies are included, but those extensions MUST be exactly as they appear in the trigger resource.

Value: Array of GenericTriggerExtension objects, where each extension object is copied from the "extensions" array values in the trigger resource.

Mandatory-to-Specify: No. The "extensions" array SHOULD be used only if the error relates to extension objects. Property omission should be interpreted as "the error is not related to any extension".

Name: cdn-id
Description: The CDN PID of the CDN where the error occurred. The "cdn-id" property is used by the originating uCDN or by the propagating dCDN to distinguish in which CDN the error occurred.

Value: A non-empty JSON string, where the string is a CDN PID as defined in Section 4.4.3

Mandatory-to-Specify: Yes. The dCDN may use its own CDN PID if it does not want to expose the PIDs of its dCDNs.

Name: objects
Description: List of objects that failed to be processed during trigger execution.

Value: An array of ObjectList (Section 4.4.2) objects. The dCDN SHOULD provide the list of objects that it failed to process during trigger execution with Section 4.1.2.8, provided that the dCDN advertised support for extended status (Section 5.5).

Mandatory-to-Specify: No. An empty array is allowed and is equivalent to omitting "objects" from the Error.v2 Description.

Example of a JSON-serialized Error.v2 Description object reporting a malformed HLS manifest:

```
{
  "error": "econtent",
  "description": "Failed to parse HLS object list",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/title/index.m3u8",
        "type": "hls"
      }]
   }}
  ],
  "objects": [{
    "href": "https://www.example.com/hls/title/index.m3u8",
    "type": "hls"
  }],
  "cdn": "AS64500:0"
}
```

Example of a JSON-serialized Error.v2 Description object reporting an unsupported extension object:

```
{
  "errors": [{
    "error": "eextension",
    "description": "unrecognized extension location-policy",
    "specs": [{
      "trigger-subject": "content",
      "cit-spec-type": "urls",
      "cit-spec-value": {
        "urls": [
          "https://www.example.com/a/b/c/1",
          "https://www.example.com/a/b/c/2"
        ]
      }
    }],
    "extensions": [{
      "cit-extension-type": "location-policy",
      "cit-extension-value": {
        "locations": [{
          "action": "deny",
          "footprints": [{
              "footprint-type": "countrycode",
              "footprint-value": [ "ca" ]
            }]
          }]
        }
    }],
    "cdn": "AS64500:0"
  }]
}
```

#### 4.1.6.2. Error Code

This type is used by the dCDN to report failures in trigger processing. All Error Codes MUST be registered in the IANA "CDNI CI/T Error Codes" registry (see Section 7.7). Unknown Error Codes MUST be treated as fatal errors, and the request MUST NOT be automatically retried without modification.

The following Error Codes are defined by this document and MUST be supported by an implementation of the CI/T v2 interface.

| Error Code | Description |
|---|---|
| emeta | The dCDN was unable to acquire and/or does not have metadata required to fulfill the request. |
| econtent | The dCDN was unable to acquire content (CI/T "preposition" commands only). |
| eperm | The uCDN does not have permission to create the trigger as requested(for example, the data is owned by another CDN). |
| ereject | The dCDN is not willing to process the trigger (for example, a "preposition" request for content at a time when the dCDN would not accept Request Routing requests from the uCDN). |
| ecdn | An internal error in the dCDN or one of its dCDNs. |
| ecancelled | The uCDN cancelled the request. |
| eunsupported | The trigger resource used an "action type" that is not supported by the dCDN. No action was taken by the dCDN other than to create a trigger in a "failed" state. |
| espec | An error occurred while parsing a generic trigger spec, or that the specific trigger spec is not supported by the CDN. |
| esubject | An error occurred while parsing a trigger subject, or that the specific trigger subject is not supported by the CDN. |
| eextension | An error occurred while parsing a generic trigger extension, or that the specific extension is not supported by the CDN. |

Table 9

## 4.2. Trigger Index Resource

As described in Section 2.1, the dCDN maintains RESTful trigger resources that represent actions ("triggers") requested by the uCDN for execution by the dCDN.

The trigger index resource maintains references to all trigger collection resources that can be used to retrieve triggers. The dCDN MUST create the trigger index resource when it first instantiates the CI/T interface for a uCDN, before any trigger resources are created. The dCDN MUST also create the unfiltered trigger collection containing all triggers, as well as trigger collections for each trigger state, and include references to these collections in the top-level trigger index resource. The dCDN MUST NOT remove the trigger index resource, the unfiltered trigger collection, or the state-based trigger collections once they have been created, as long as it continues to offer CI/T services to the uCDN.

The dCDN MUST update both the unfiltered trigger collection and the state-based trigger collections when triggers are created, deleted, or their state changes.

If triggers include labels, the dCDN MUST create and maintain a trigger collection resource for each label value used in any trigger resources, and update the trigger index to keep references to label-based trigger collections current. The dCDN MUST remove a label-based trigger collection when no triggers remain that use the corresponding label value.

As a top-level resource, the trigger index also includes global dCDN attributes, such as the "staleresourcetime" attribute, which regulates the expiration of completed triggers, and the "cdn-id" attribute, which indicates the CDN PID of the dCDN.

The trigger index resource representation MUST use a MIME media type of "application/cdni; ptype=ci-trigger-index.v2".

A trigger index is encoded as a JSON object containing the following name/value pairs:

Name: collections
Description: Array of Trigger Collection View (Section 4.3.1) objects.

Value: An array of JSON-encoded Trigger Collection View objects, one for each existing trigger collection resource. This includes the unfiltered trigger collection, per-state trigger collections (one for each trigger state as specified in Section 4.1.5), and any per-label trigger collections, if they exist.

Mandatory-to-Specify: Yes.

Name: staleresourcetime

Description: The length of time for which the dCDN guarantees to keep a completed trigger resource. After this time, the dCDN SHOULD delete the trigger resource and all references to it from the collection.

Value: A JSON number, which must be a positive integer, representing time in seconds.

Mandatory-to-Specify: Yes.

Name: cdn-id
Description: The dCDN PID.

Value: A JSON string, dCDN’s PID, as defined in Section 4.4.3.

Mandatory-to-Specify: No.

## 4.3. Trigger Collection Resource

The collection of trigger resources represents triggers created by the dCDN, optionally filtered by a parameter.

Trigger resources in a collection are usually represented using their unique URIs. Note that the collection may refer to CI/T Resources from several versions of CI/T objects, i.e., a subsequent call for the retrieval of the relevant trigger resource may provide objects of various MIME media types: ci-trigger-status as defined in [RFC8007], ci-trigger.v2 defined in this document, or objects of future CI/T objects versions, based on the version of the JSON object used to create the trigger.

To allow the uCDN to check the status of multiple triggers in a single request, the dCDN MAY maintain optional representations of the trigger collection, which contain a subset of all triggers, filtered using a parameter. These filtered collection representations are "optional-to-implement", but if they are implemented, the dCDN MUST include links to them in the trigger collection resource.

All trigger collection representations MUST use a MIME media type of "application/cdni; ptype=ci-trigger-collection.v2".

A trigger collection is encoded as a JSON object containing the following name/value pairs:

Name: trigger-urls
Description: Links to trigger resources in the collection.

Value: A JSON array of zero or more URLs, represented as JSON strings.

Mandatory-to-Specify: Yes.

Name: trigger-objects
Description: Array of all triggers in the collection. Should be returned only when an extended trigger collection view is requested as described in Section 3.4.1.1.

Value: An array of JSON-encoded trigger resources.

Mandatory-to-Specify: No. The "trigger-objects" attribute SHOULD only be used by the dCDN that supports and advertises the appropriate extended status for trigger collections (see Section 5.5 for details).

Name: filter-type
Description: Indicates the type of filter applied to select the triggers in the collection.

Value: One of "state" or "label".

Mandatory-to-Specify: No. The "filter-type" attribute MUST be used only in per-state and per-label trigger collections. If omitted or empty, the trigger collection MUST include all existing triggers.

Name: filter-value
Description: Specifies the filter value used to select the triggers in the collection.

Value: For per-label trigger collections, this attribute MUST contain the corresponding label value. For per-state trigger collections, it MUST contain the trigger state shared by all triggers in the collection.

Mandatory-to-Specify: No. The "filter-value" attribute MUST be used only in per-state and per-label trigger collections. If omitted or empty, the trigger collection MUST include all existing triggers.

#### 4.3.1. Trigger Collection View

The Trigger Collection View is used to provide a brief description of the trigger collection resource within the trigger index. It is encoded as a JSON object containing the following name/value pairs:

Name: filter-type
Description: Indicates the type of filter applied to select the triggers in the described trigger collection.

Value: One of "state" or "label".

Mandatory-to-Specify: No. The "filter-type" attribute MUST be used to describe only per-state and per-label trigger collections. If omitted or empty, the trigger collection view MUST point to the trigger collection of all triggers.

Name: filter-value
Description: Specifies the filter value used to select the triggers in the collection.

Value: For per-label trigger collections, this attribute MUST contain the corresponding label value. For per-state trigger collections, it MUST contain the trigger state shared by all triggers in the collection.

Mandatory-to-Specify: No. The "filter-value" attribute MUST be used to describe only per-state and per-label trigger collections. If omitted or empty, the trigger collection view MUST point to the trigger collection of all triggers.

Name: collection-uri
Description: URI of the trigger collection.

Value: A URI represented as a JSON string.

Mandatory-to-Specify: Yes.

### 4.4. Other CI/T Objects and Properties

This section describes common CI/T objects, which are used as part of the specification of several other CI/T objects, and their encodings.

#### 4.4.1. URL Type

This type is used by the uCDN to indicate how to interpret URLs referenced by trigger specs that use URLs, such as Section 4.1.2.4, Section 4.1.2.6, Section 4.1.2.7, and Section 4.1.2.8.

One option is for the uCDN to use "published" URLs. Published URLs are the URLs used by end users. When processing a trigger that uses this URL type, the dCDN MUST be able to match the URLs with metadata objects provided by the uCDN. When this is not the case, the dCDN MUST return the error code "emeta".

When processing published URLs in "preposition" trigger action, the dCDN MUST invoke processing of metadata objects it would have invoked in content acquisition to satisfy an end-user request, e.g., SourceMetadata (see Section 4.2.1 of [RFC8006]).

Another type of URL in common use is a "private" URL. A private URL is based on cache keys that are dynamically constructed via lightweight processing of various properties of the HTTP request and/or response. As an example, an origin might specify a cache key as a value returned in a specific HTTP response header.

As an example, the uCDN may prefer to use such private URLs in "purge" or "invalidate" trigger actions to simplify processing.

The dCDNs implementing the CI/T interface MUST support the "published" URL type. The dCDN MAY support the additional "private" URL type. In this case, the dCDN SHOULD advertise the private URL type support via FCI using Section 5.4. If the private URL is not supported by the dCDN, it SHOULD reject the trigger creation request using "eunsupported" Error Code. If both URL types are supported by the dCDN, the uCDN MUST use only one URL type in each trigger.

The following URL types are defined by this document and MUST be supported by the implementation of the CI/T interface:

| URL Type | Description |
|---|---|
| published | Published URL used by end users to access content |
| private | Private URLs used by the dCDN to look up content objects in cache |

Table 10

### 4.4.2. ObjectList

ObjectList is a metadata object describing lists of objects that can be used in the context of CI/T v2 trigger specs, trigger status resources, or other contexts as required. The ObjectList object can either embed the lists of objects or point to external URL(s) that hold such lists. ObjectList allows the specification of an object list type, providing instructions on the interpretation of the object list format.

ObjectLists MAY be recursive, i.e., including references to secondary manifests, including references to HLS, MPEG-DASH, or MSS manifests as well as additional JSON-encoded ObjectLists, etc. The party consuming the object list MUST parse all recursions based on the object list type property. When doing so, the consuming party SHOULD also detect potential loops when the descendant ObjectList points back to its parent ObjectList.

In the case of the uCDN accessing ObjectList objects referencing external URLs published by the dCDN, both parties SHOULD comply with the CI/T interface security requirements (see Section 8.1 for details). When the dCDN accesses external URLs referenced by ObjectLists supplied by the uCDN, for example as part of trigger spec, the dCDN MUST match these URLs with source metadata objects, published by the uCDN, such as SourceMetadata objects specified in Section 4.2.1 of [RFC8006], and use these metadata objects for content acquisition if a match was found.

ObjectLists MAY combine regular objects and secondary ObjectLists in the same object. Please note that when embedding non-JSON object lists directly in ObjectList, absolute URLs MUST be provided at all times and the text SHOULD be encoded per the JSON grammar specification [ECMA404], including explicit newline encoding. When the uCDN accesses ObjectList metadata resources published by the dCDN, the same interface authentication and authorization requirements would apply, as when accessing the interface itself.

The ObjectList properties are as follows:

Name: data
Description: List of objects in one of the recognized formats.

Value: JSON string.

Mandatory-to-Specify: No. Either "data" or "href" MUST be set.

Name: href
Description: URL pointing to an external object list or object in one of the recognized formats.

Value: A URL represented as a JSON string.

Mandatory-to-Specify: No. Either "data" or "href" MUST be set.

Name: type

Description: Object list type to be used when parsing and interpreting this object list. By default, each record in the list is assumed to represent a content or metadata object that does not require additional processing.

Value: ObjectListType (see Section 4.4.2.1).

Mandatory-to-Specify: Yes.

ObjectList is encoded as an array of per-object records in JSON format as follows:

External HLS manifest:

```
[
  {
    "href": "https://example.com/hls/a36f764e/index.m3u8",
    "type": "hls"
  }
]
```

External object list in text format:

```
[
  {
    "href": "https://example.com/hls/35cdc008/assets",
    "type": "text"
  }
]
```

List of external manifests and objects of mixed types:

```
[
  {
    "href": "https://example.com/hls/35cdc008/index.m3u8",
    "type": "hls"
  },
  {
    "href": "https://example.com/dash/35cdc008/main.mpd",
    "type": "dash"
  },
  {
    "href": "https://example.com/dash/35cdc008/files.json",
    "type": "json"
  }
]
```

Embedded JSON-encoded object list:

```
[
  {
    "data": [
      {
        "href": "https://example.com/hls/35cdc008/index.m3u8",
        "type": "hls"
      },
      {
        "href": "https://example.com/dash/35cdc008/main.mpd",
        "type": "dash"
      },
      {
        "href": "https://example.com/img/35cdc008/thumb-l.jpg",
        "size": 10260
      },
      {
        "href": "https://example.com/img/35cdc008/thumb-s.jpg",
        "size": 1453
      }
    ],
    "type": "json"
  }
]
```

Embedded HLS manifest:

Please note that some lines in the example are wrapped for clarity.

```
[
  {
    "data": "#EXTM3U\n
#EXT-X-STREAM-INF:BANDWIDTH=150000,RESOLUTION=416x234,
  CODECS=\"avc1.42e00a,mp4a.40.2\"\n
http://example.com/low/index.m3u8\n
#EXT-X-STREAM-INF:BANDWIDTH=240000,RESOLUTION=416x234,
  CODECS=\"avc1.42e00a,mp4a.40.2\"\n
http://example.com/lo_mid/index.m3u8\n
#EXT-X-STREAM-INF:BANDWIDTH=440000,RESOLUTION=416x234,
  CODECS=\"avc1.42e00a,mp4a.40.2\"\n
http://example.com/hi_mid/index.m3u8\n
#EXT-X-STREAM-INF:BANDWIDTH=640000,RESOLUTION=640x360,
  CODECS=\"avc1.42e00a,mp4a.40.2\"\n
http://example.com/high/index.m3u8\n
#EXT-X-STREAM-INF:BANDWIDTH=64000,
  CODECS=\"mp4a.40.5\"\n
http://example.com/high/index.m3u8\n",
    "type": "hls"
  }
]
```

#### 4.4.2.1. ObjectList Type

ObjectListType objects are used to specify the registered type of ObjectList objects (see Section 7.5), used in trigger spec, trigger objects, and Error.v2 Description objects.

The following table defines the initial ObjectListType JSON string values

| JSON string | Description | Protocol Specification |
|---|---|---|
| hls | HTTP Live Streaming | RFC 8216 [RFC8216] |
| mss | Microsoft Smooth Streaming | MSS [MSS] |
| dash | Dynamic Adaptive Streaming over HTTP (MPEG-DASH) | MPEG-DASH [MPEG-DASH] |
| json | JSON-serialized object list | JSON (Section 4.4.2.2) |
| text | Object list in text format | Text (Section 4.4.2.3) |

Table 11

If the support for the ObjectList Spec (Section 4.1.2.8) is implemented, the dCDN MUST support at least one of the ObjectList types.

#### 4.4.2.2. JSON Serialized Object List

This ObjectList type specifies a collection of objects encoded in JSON format, where each entry is encoded as an ObjectEntry (Section 4.4.2.4) object. The entries in the JSON object list MAY have an object list type specified, allowing for a recursive object list structure.

#### 4.4.2.3. Text Object List

Unlike the JSON Serialized Object List (Section 4.4.2.2), the text-based object list will not support a recursive object list structure, and every object specified in it SHOULD be acted upon without additional processing.

#### 4.4.2.4. ObjectEntry

ObjectEntry is a metadata object describing an object and its associated metadata, to be used in JSON-encoded ObjectList (Section 4.4.2) objects.

The ObjectEntry properties are as follows:

Name: href
Description: Object URL

Value: A URL represented as a JSON string.

Mandatory-to-Specify: Yes.

Name: type
Description: ObjectList type to be used when processing this object. By default, the ObjectEntry object is assumed to represent an object and does not require additional processing.

Value: ObjectListType (see Section 4.4.2.1).

Mandatory-to-Specify: No.

Name: size
Description: Object size, in bytes. Can be used to decide to download the object based on size. For example, the dCDN may ignore objects that are too small or too large.

Value: Integer.

Mandatory-to-Specify: No.

The following is an example of JSON-serialized ObjectEntry objects:

```
[
  {
    "href": "https://example.com/hls/35cdc008/index.m3u8",
    "type": "hls"
  },
  {
    "href": "https://example.com/dash/35cdc008/main.mpd",
    "type": "dash"
  },
  {
    "href": "https://example.com/img/35cdc008/thumb-l.jpg",
    "size": 102600
  },
  {
    "href": "https://example.com/img/35cdc008/thumb-s.jpg",
    "size": 14535
  }
]
```

#### 4.4.3. CDN Provider ID

The CDN PID consists of the two characters "AS" followed by the CDN provider's Autonomous System number [RFC1930], then a colon (":") and an additional qualifier that is used to guarantee uniqueness in case a particular AS has multiple independent CDNs deployed -- for example, "AS64496:0".

If the CDN provider has multiple ASes, the same AS number SHOULD be used in all messages from that CDN provider, unless there are multiple distinct CDNs.

If the CDNI Request Routing Redirection interface (RI) described in [RFC7975] is implemented by the dCDN, the CI/T interface and the RI SHOULD use the same CDN PID.

## 5. Footprint and Capabilities

This section covers the FCI objects required for the advertisement of the specs, extensions, and properties introduced in this document.

### 5.1. CI/T Endpoint Capability Object

The CI/T trigger endpoint capability object is used to advertise one or more CI/T interface endpoints along with CI/T interface versions supported by these endpoints. The capability type is "FCI.CITEndpoints". Version 1, as originally defined in [RFC8007], is the default if this capability is not explicitly declared.

A CI/T Endpoints capability object is encoded as an array of JSON objects containing the following name/value pairs:

Name: trigger-endpoint-uri
Description: CI/T endpoint URI

Value: A URL represented as a JSON string.

Mandatory-to-Specify: Yes.

Name: trigger-versions
Description: A list of CI/T versions supported by the trigger endpoint.

Value: An array of JSON strings. Valid values include "v2", corresponding to this version of the interface, and "v1", corresponding to [RFC8007].

Mandatory-to-Specify: Yes.

Name: trigger-subjects
Description: Array of trigger subjects supported by the trigger endpoint.

Value: An array of Strings containing the type of the subject matching the cit-spec-value property, such as "content" or "metadata" as defined in Section 4.1.2.2.

Mandatory-to-Specify: No. A missing or empty "trigger-subjects" list means that all trigger subjects are supported by the endpoint. The dCDN SHOULD advertise only one endpoint for every trigger subject and CI/T interface version pair. If more than one interface endpoint supports the same trigger subject and CI/T interface version (e.g., CI/T version 2 interface for content objects), the uCDN SHOULD be able to use any of the advertised CI/T interface endpoints interchangeably.

#### 5.1.1. CI/T Endpoints Capability Object Serialization

The following example shows the serialization of a CI/T Endpoints Capability object for a dCDN that supports version 2 of the CI/T interface for content trigger subjects and version 1 for metadata trigger subjects, with one metadata endpoint for both the US and Brazil, and two separate content endpoints for the two countries.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.CITEndpoints",
      "capability-value": {
        "trigger-endpoint-uri":
          "https://dcdn.example/configuration/",
        "trigger-versions": [ "v1" ],
        "trigger-subjects": "metadata"
      },
      "footprints": {
        "footprint-type": "countrycode",
        "footprint-value": [ "us", "br" ]
      }
    },
    {
      "capability-type": "FCI.CITEndpoints",
      "capability-value": {
        "trigger-endpoint-uri":
          "https://dcdn.example/cache-management-us/",
        "trigger-versions": [ "v2" ],
        "trigger-subjects": "content"
      },
      "footprints": {
        "footprint-type": "countrycode",
        "footprint-value": [ "us" ]
      }
    },
    {
      "capability-type": "FCI.CITEndpoints",
      "capability-value": {
        "trigger-endpoint-uri":
          "https://dcdn.example/cache-management-br/",
        "trigger-versions": [ "v2" ],
        "trigger-subjects": "content"
      },
      "footprints": {
        "footprint-type": "countrycode",
        "footprint-value": [ "br" ]
      }
    }
  ]
}
```

## 5.2. CI/T Trigger Scope Capability Object

The CI/T supports several trigger actions for different trigger subjects as defined in Section 4.1.1 and Section 4.1.2.2. Additional actions, as well as subjects, may be defined in the future. The trigger scope capability object is used to indicate support for a trigger action for a subject. It further specifies the trigger generic spec types that may be used for selecting the targets to which the triggers apply, along with the supported trigger generic extension types.

All supported combinations of trigger actions and trigger subjects MUST be explicitly advertised, with one FCI.CITScope object provided for each combination.

The "trigger-scope-capability" object matches the "FCI.CITScope" capability type and is encoded as a JSON object containing the following name/value pairs:

Name: trigger-action
Description: The supported CDNI CI/T trigger action.

Value: A string corresponding to an entry from the "CDNI CI/T Trigger Types" registry Section 7.2, which corresponds to a CDNI CI/T trigger action.

Mandatory-to-Specify: Yes.

Name: trigger-subject
Description: The supported CDNI CI/T trigger subject.

Value: A string corresponding to an entry from the "CDNI CI/T Trigger Subjects" registry Section 7.4, which corresponds to a CDNI CI/T trigger subject.

Mandatory-to-Specify: Yes.

Name: trigger-specs
Description: A list of supported CDNI CI/T GenericSpecObject types for trigger action and subject.

Value: List of JSON strings corresponding to entries from the "CDNI CI/T Trigger Specs" registry Section 7.3, which correspond to CDNI CI/T GenericSpecObject objects.

Mandatory-to-Specify: No. The default in case of a missing or an empty list MUST be interpreted as "no GenericSpecObject types supported". A non-empty list MUST be interpreted as containing "the only GenericSpecObject types that are supported".

Name: trigger-extensions
Description: A list of supported CDNI CI/T GenericExtensionObject types for trigger action and subject.

Value: List of JSON strings corresponding to entries from the "CDNI CI/T Trigger Extension" registry Section 7.6, which corresponds to a CDNI CI/T GenericExtensionObject object.

Mandatory-to-Specify: No. The default in case of a missing or an empty list MUST be interpreted as "no GenericExtensionObject types are supported". A non-empty list MUST be interpreted as containing "the only GenericExtensionObject types that are supported".

#### 5.2.1. CI/T Trigger Scope Capability Object Serialization

The following shows an example of a JSON-serialized CI/T Trigger Scope Capability objects serialization for the dCDN that supports the preposition and invalidation of content, using "urls" and "ccids" generic spec types, with "time-policy" but only for the "preposition" action. Note that in this example, purge is not supported, and no actions involving metadata are supported either.

```
{
     "capabilities": [
        {
          "capability-type": "FCI.CITScope",
          "capability-value": {
             "trigger-action": "preposition",
             "trigger-subject": "content",
             "trigger-specs": [ "urls", "ccids" ],
             "trigger-extensions": [ "time-policy" ]
          },
          "footprints": {
            "footprint-type": "countrycode",
            "footprint-value": [ "us" ]
          }
        },
        {
          "capability-type": "FCI.CITScope",
          "capability-value": {
             "trigger-action": "invalidate",
             "trigger-subject": "content",
             "trigger-specs": [ "urls", "ccids" ]
          },
          "footprints": {
            "footprint-type": "countrycode",
            "footprint-value": [ "us" ]
          }
        }
     ]
}
}
```

### 5.3. CI/T Object List Type Capability Object

Given an object list being supported by the dCDN, the CI/T Object List Type capability object is used to indicate support for one or more Object List types listed in Section 7.5 by the type property of the "ObjectList" object. The capability type is "FCI.CITObjectListType".

Name: object-list-types
Description: A list of supported ObjectList types.

Value: An array of JSON strings representing ObjectListTypes (Section 4.4.2.1).

Mandatory-to-Specify: No. A missing or an empty list MUST be interpreted as no ObjectList types are supported.

#### 5.3.1. CI/T Object List Type Capability Object Serialization

The following shows an example of a JSON-serialized CI/T Object List Type Capability object serialization for the dCDN that supports "hls", "dash", and "json", in the US only.

```
{
  "capabilities": [{
    "capability-type": "FCI.CITObjectListType",
    "capability-value": {
      "object-list-types": [ "hls", "dash", "json" ]
    },
    "footprints": {
      "footprint-type": "countrycode",
      "footprint-value": [ "us" ]
    }
  }]
}
```

### 5.4. CI/T Private URL Capability Object

The CI/T Private URL capability object is used to indicate support for operations on private URLs (see Section 4.4.1 for details). The capability type is "FCI.CITPrivateUrlType".

Name: private-url-type-support

Description: Indicate whether private URL type is supported by the dCDN.

Value: Boolean.

Mandatory-to-Specify: No. A missing or an empty attribute MUST be interpreted as no support for private URLs.

#### 5.4.1. CI/T Private URL Type Capability Object Serialization

The following shows an example of a JSON-serialized CI/T Private URL Type Capability object serialization for the dCDN that supports the private URL type in URL-based trigger spec types.

```
{
  "capabilities": [{
    "capability-type": "FCI.CITPrivateUrlType",
    "capability-value": { "private-url-type-support": true },
    "footprints": {
      "footprint-type": "countrycode",
      "footprint-value": [ "us" ]
    }
  }]
}
```

### 5.5. CI/T Extended Status Capability Object

The CI/T extended trigger status capability object is used to indicate support for extended trigger status. The extended trigger status is returned upon the uCDN request and includes:

* "objects" attribute in the trigger object

* "objects" attribute in Error.v2 description object

* "trigger-objects" attribute in the trigger collection object

The capability type is "FCI.CITExtendedStatus".

Name: extended-status-objects
Description: List of CI/T objects that support extended attributes.

Value: An array of JSON strings listing CI/T objects.

Mandatory-to-Specify: No. By default, in case of a missing or an empty list, no extended attribute objects are supported.

#### 5.5.1. CI/T Private URL Type Capability Object Serialization

The following shows an example of a JSON-serialized CI/T Extended Status Type Capability object serialization for the dCDN that supports extended status in trigger, Error.v2 description, and trigger collections objects.

```
{
  "capabilities": [{
    "capability-type": "FCI.CITExtendedStatus",
    "capability-value": {
      "extended-status-objects": [
        "trigger-state",
        "error-v2-description",
        "trigger-collection"
      ]
    },
    "footprints": {
      "footprint-type": "countrycode",
      "footprint-value": [ "us" ]
    }
  }]
}
```

# 6. Examples

This section provides examples of using the CI/T interface and its features.

The discovery of the CI/T interface is out of the scope of this document. In an implementation, all CI/T URLs are under the control of the dCDN. The uCDN MUST NOT attempt to ascribe any meaning to individual elements of the path.

In examples in this section, the root URI "https://dcdn.example/cit/" is used as the location of the trigger collection resource, and the PID of the uCDN is "AS64496:1".

## 6.1. Creating Triggers

### 6.1.1. Preposition

Below is an example of a "preposition" trigger creation. The uCDN sends HTTP POST request to the trigger collection URI with the trigger representation in the request body.

Note that pattern-based or label-based specs like "uri-pattern-match", "uri-regex-match" and "ccids" are not allowed to be used with "preposition" trigger action, where the dCDN MUST have a clear list of objects to obtain.

REQUEST:

```
POST /cit HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger.v2
Content-Length: 622

{
  "action": "preposition",
  "specs": [
    {
      "trigger-subject": "metadata",
      "cit-spec-type": "urls",
      "cit-spec-value": {
      "urls": [ "https://metadata.example.com/a/b/c" ]
           }
    },
    {
      "trigger-subject": "content",
      "cit-spec-type": "urls",
      "cit-spec-value": {
        "urls": [
          "https://www.example.com/a/b/c/1",
          "https://www.example.com/a/b/c/2",
          "https://www.example.com/a/b/c/3",
          "https://www.example.com/a/b/c/4"
        ]
      }
    }
  ],
  "cdn-path": [ "AS64496:1" ]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sun, 27 Oct 2024 08:48:10 GMT
Content-Length: 710
Content-Type: application/cdni; ptype=ci-trigger.v2
Location: https://dcdn.example/cit/3f2d259d-a980-4742-beeb-9392a58129f5
Server: example-server/0.1

{
  "ctime": 1730119690,
  "etime": 1730119750,
```

```
   "mtime": 1730119690,
   "state": "pending",
   "action": "preposition",
   "specs": [
     {
       "trigger-subject": "metadata",
       "cit-spec-type": "urls",
       "cit-spec-value": {
         "urls": [ "https://metadata.example.com/a/b/c" ]
       }
     },
     {
       "trigger-subject": "content",
       "cit-spec-type": "urls",
       "cit-spec-value": {
         "urls": [
           "https://www.example.com/a/b/c/1",
           "https://www.example.com/a/b/c/2",
           "https://www.example.com/a/b/c/3",
           "https://www.example.com/a/b/c/4"
         ]
       }
     }
   ],
   "cdn-path": [ "AS64496:1" ]
 }
```

#### 6.1.2. Invalidate

Below is an example of a CI/T "invalidate" trigger creation. This trigger instructs the dCDN to revalidate:

* the metadata objects with URLs prefixed by "https://metadata.example.com/a/b/" using case-insensitive matching

* a single content object identified by the URL "https://www.example.com/a/index.html"

* the content objects with URLs prefixed by "https://www.example.com/a/b/" using case-sensitive matching

REQUEST:

```
POST /cit HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

```
Content-Type: application/cdni; ptype=ci-trigger.v2
Content-Length: 783

{
  "action": "invalidate",
  "specs": [
     {
        "trigger-subject": "metadata",
        "cit-spec-type": "uri-pattern-match",
        "cit-spec-value": {
           "pattern": "https://metadata.example.com/a/b/*"
        }
     },
     {
        "trigger-subject": "content",
        "cit-spec-type": "urls",
        "cit-spec-value": {
           "urls": [
              "https://www.example.com/a/index.html"
           ]
        }
     },
     {
        "trigger-subject": "content",
        "cit-spec-type": "uri-pattern-match",
        "cit-spec-value": {
           "pattern": "https://www.example.com/a/b/*",
           "case-sensitive": true
        }
     }
  ],
  "cdn-path": [ "AS64496:1" ]
}

RESPONSE:

HTTP/1.1 201 Created
Date: Sun, 27 Oct 2024 08:48:11 GMT
Content-Length: 807
Content-Type: application/cdni; ptype=ci-trigger.v2
Location: https://dcdn.example/cit/327df5b8-1df8-4cff-92f8-fda27774c171
Server: example-server/0.1

{
  "ctime": 1730119691,
  "etime": 1730119751,
  "mtime": 1730119691,
  "state": "pending",
```

```
   "action": "invalidate",
   "specs": [
     {
       "trigger-subject": "metadata",
       "cit-spec-type": "uri-pattern-match",
       "cit-spec-value": {
         "pattern": "https://metadata.example.com/a/b/*"
       }
     },
     {
       "trigger-subject": "content",
       "cit-spec-type": "urls",
       "cit-spec-value": {
         "urls": [ "https://www.example.com/a/index.html" ]
       }
     },
     {
       "trigger-subject": "content",
       "cit-spec-type": "uri-pattern-match",
       "cit-spec-value": {
         "pattern": "https://www.example.com/a/b/*",
         "case-sensitive": true
       }
     }
   ],
   "cdn-path": [ "AS64496:1" ]
 }
```

#### 6.1.3. Invalidation with Regex

In the following example, a CI/T "invalidate" trigger uses the Regex property to specify the range of content objects for invalidation, the trigger is rejected by the dCDN due to regex complexity, and an appropriate error is reflected in the response.

Please note that some lines in the example are wrapped for clarity.

```
 REQUEST:

 POST /cit HTTP/1.1
 User-Agent: example-user-agent/0.1
 Host: dcdn.example
 Accept: */*
 Content-Type: application/cdni; ptype=ci-trigger.v2
 Content-Length: 392

 {
   "action": "invalidate",
```

```
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "uri-regex-match",
    "cit-spec-value": {
      "regex": "^(https:\\/\\/video\\.example\\.com)\\/
        ([a-z])\\/movie1\\/([1-7])\\/*(index.m3u8|\\d{3}.ts)$",
      "case-sensitive": true,
      "match-query-string": false
    }
  }],
  "cdn-path": [ "AS64496:0" ]
}

RESPONSE:

HTTP/1.1 201 Created
Date: Sun, 27 Oct 2024 08:48:12 GMT
Content-Length: 960
Content-Type: application/cdni; ptype=ci-trigger.v2
Location: https://dcdn.example/cit/991b9fb9-d0be-4d05-be06-64c0e5c5a5f9
Server: example-server/0.1

{
  "errors": [{
    "specs": [{
      "trigger-subject": "content",
      "cit-spec-type": "uri-regex-match",
      "cit-spec-value": {
      "regex": "^(https:\\/\\/video\\.example\\.com)\\/([a-z])\
        \/movie1\\/([1-7])\\/*(index.m3u8|\\d{3}.ts)$",
        "case-sensitive": true,
        "match-query-string": false
      }
    }],
    "description": "The dCDN rejected a regex due to complexity",
    "error": "ereject",
    "cdn": "AS64500:0"
  }],
  "ctime": 1730119692,
  "etime": 1730119692,
  "mtime": 1730119692,
  "state": "failed",
  "action": "invalidate",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "uri-regex-match",
    "cit-spec-value": {
      "regex": "^(https:\\/\\/video\\.example\\.com)\\/([a-z])\
```

```
        \/movie1\\/([1-7])\\/*(index.m3u8|\\d{3}.ts)$",
      "case-sensitive": true,
      "match-query-string": false
    }
  }],
  "cdn-path": [ "AS64496:0" ]
}
```

#### 6.1.4. Preposition with ObjectLists

In the following example, a CI/T "preposition" trigger uses the ObjectList property to specify the full media library of a specific content. The command fails due to object list parse error and an appropriate error is reflected in the response.

REQUEST:

```
POST /cit HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger.v2
Content-Length: 328

{
  "action": "preposition",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/title/index.m3u8",
        "type": "hls"
      }]
    }
  }],
  "cdn-path": [ "AS64496:0" ]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sun, 27 Oct 2024 08:48:13 GMT
Content-Length: 829
Content-Type: application/cdni; ptype=ci-trigger.v2
Location: https://dcdn.example/cit/86633e6e-d2da-4185-a285-b3d087a5d711
Server: example-server/0.1
```

```
{
  "errors": [{
    "specs": [{
      "trigger-subject": "content",
      "cit-spec-type": "content-objectlist",
      "cit-spec-value": {
        "objects": [{
          "href": "https://www.example.com/hls/title/index.m3u8",
          "type": "hls"
        }}
    }],
    "description": "The dCDN was not able to parse the object list",
    "error": "econtent",
    "cdn": "AS64500:0"
  }],
  "ctime": 1730119693,
  "etime": 1730119693,
  "mtime": 1730119693,
  "state": "failed",
  "action": "preposition",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/title/index.m3u8",
        "type": "hls"
      }]
    }
  }],
  "cdn-path": [ "AS64496:0" ]
}
```


## 6.2. Changing, Cancelling and Deleting Triggers

### 6.2.1. Modifying Triggers

The uCDN can modify triggers while they are in a "pending" state. One example of this might be to adjust a trigger's "specs" and/or "labels" attributes. In the below example, the uCDN updates a trigger created earlier by removing the metadata portion of the trigger spec and adding trigger labels. The dCDN responds with a 200 ("OK") response containing the updated trigger representation.

REQUEST:

```
POST /cit/3f2d259d-a980-4742-beeb-9392a58129f5 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger.v2
Content-Length: 401

{
  "specs": [
    {
      "trigger-subject": "content",
      "cit-spec-type": "urls",
      "cit-spec-value": {
        "urls": [
          "https://www.example.com/d/e/f/1",
          "https://www.example.com/d/e/f/2",
          "https://www.example.com/d/e/f/3",
          "https://www.example.com/d/e/f/4"
        ]
      }
    }
  ],
  "labels": [
    "type=video"
  ]
}
```

RESPONSE:

```
HTTP/1.1 200 OK
Date: Sun, 27 Oct 2024 08:48:14 GMT
Content-Length: 520
Content-Type: application/cdni; ptype=ci-trigger.v2
Server: example-server/0.1

{
  "ctime": 1730119694,
  "etime": 1730119754,
  "mtime": 1730119694,
  "state": "pending",
  "action": "preposition",

  "specs": [
    {
      "trigger-subject": "content",
      "cit-spec-type": "urls",
```

```
        "cit-spec-value": {
          "urls": [
            "https://www.example.com/d/e/f/1",
            "https://www.example.com/d/e/f/2",
            "https://www.example.com/d/e/f/3",
            "https://www.example.com/d/e/f/4"
          ]
        }
      }
    ],
    "labels": [
      "type=video"
    ]
  }
```

#### 6.2.2. Cancelling Triggers

The uCDN can cancel triggers that are not in a terminal state by requesting to update the trigger state to "cancelled". In case of asynchronous processing, the dCDN will respond by setting the trigger state to "cancelling" and update it "cancelled" when the cancellation is complete.

```
REQUEST:

POST /cit/3f2d259d-a980-4742-beeb-9392a58129f5 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger.v2
Content-Length: 27

{
  "state": "cancelled"
}

RESPONSE:

HTTP/1.1 200 OK
Date: Sun, 27 Oct 2024 08:48:15 GMT
Content-Length: 523
Content-Type: application/cdni; ptype=ci-trigger.v2
Server: example-server/0.1

{
  "ctime": 1730119695,
  "etime": 1730119755,
  "mtime": 1730119695,
  "state": "cancelling",
  "action": "preposition",

  "specs": [
    {
      "trigger-subject": "content",
      "cit-spec-type": "urls",
      "cit-spec-value": {
        "urls": [
          "https://www.example.com/d/e/f/1",
          "https://www.example.com/d/e/f/2",
          "https://www.example.com/d/e/f/3",
          "https://www.example.com/d/e/f/4"
        ]
      }
    }
  ],
  "labels": [
    "type=video"
  ]
}
```

#### 6.2.3. Deleting Triggers

The uCDN can delete completed and failed triggers to reduce the size of the collections, as described in Section 3.5. For example, to delete the "preposition" trigger from earlier examples:

REQUEST:

```
DELETE /cit/3f2d259d-a980-4742-beeb-9392a58129f5 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

RESPONSE:

```
HTTP/1.1 204 No Content
Date: Sun, 27 Oct 2024 08:48:16 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: example-server/0.1
```

### 6.3. Examining Trigger Status

Once triggers have been created, the uCDN can check their status as shown in the following examples.

#### 6.3.1. Trigger Index

The uCDN can fetch all active trigger collections, representing all existing trigger resources.

REQUEST:

```
GET /cit HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 341
Expires: Sun, 27 Oct 2024 08:49:18 GMT
Server: example-server/0.1
ETag: "936094426920308378"
Last-Modified: Sun, 27 Oct 2024 08:40:17 GMT
Cache-Control: max-age=60
Date: Sun, 27 Oct 2024 08:48:18 GMT
```

```
Content-Type: application/cdni; ptype=ci-trigger-index.v2

{
  "cdn-id": "AS64496:0",
  "staleresourcetime": 86400,
  "collections": [
    {
      "uri": "/cit/all"
    },
    {
      "filter-type": "state",
      "filter-value": "pending",
      "uri": "/cit/state/pending"
    },
    {
      "filter-type": "state",
      "filter-value": "active",
      "uri": "/cit/state/active"
    },
    {
      "filter-type": "state",
      "filter-value": "complete",
      "uri": "/cit/state/complete"
    },
    {
      "filter-type": "state",
      "filter-value": "processed",
      "uri": "/cit/state/processed"
    },
    {
      "filter-type": "state",
      "filter-value": "failed",
      "uri": "/cit/state/failed"
    },
    {
      "filter-type": "state",
      "filter-value": "cancelling",
      "uri": "/cit/state/cancelling"
    },
    {
      "filter-type": "state",
      "filter-value": "cancelled",
      "uri": "/cit/state/cancelled"
    },
    {
      "filter-type": "label",
      "filter-value": "type=video",
      "uri": "/cit/labels/type=video"
```

```
      }
    ]
  }
```

### 6.3.2. Trigger Collection

Before the dCDN starts processing the remaining trigger shown above, it will appear in the collection of pending triggers. For example:

REQUEST:

```
GET /cit/state/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 123
Expires: Sun, 27 Oct 2024 08:49:19 GMT
Server: example-server/0.1
ETag: "4331492443626270781"
Last-Modified: Sun, 27 Oct 2024 08:40:17 GMT
Cache-Control: max-age=60
Date: Sun, 27 Oct 2024 08:48:19 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection.v2
327df5b8-1df8-4cff-92f8-fda27774c171
{
  "trigger-urls": [
    "https://dcdn.example/cit/327df5b8-1df8-4cff-92f8-fda27774c171"
  ]
}
```

At this point, if no other triggers had been created, the trigger collection for failed triggers would hold the two failed triggers shown above while other trigger collecions would be empty. For example:

REQUEST:

```
GET /cit/state/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 51
Expires: Sun, 27 Oct 2024 08:49:20 GMT
Server: example-server/0.1
ETag: "7958041393922269003"
Last-Modified: Sun, 27 Oct 2024 08:48:17 GMT
Cache-Control: max-age=60
Date: Sun, 27 Oct 2024 08:48:20 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection.v2

{
  "trigger-urls": []
}
```

REQUEST:

```
GET /cit/state/failed HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 191
Expires: Sun, 27 Oct 2024 08:49:21 GMT
Server: example-server/0.1
ETag: "4331492443626270781"
Last-Modified: Sun, 27 Oct 2024 08:48:13 GMT
Cache-Control: max-age=60
Date: Sun, 27 Oct 2024 08:48:19 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection.v2

{
  "trigger-urls": [
    "https://dcdn.example/cit/991b9fb9-d0be-4d05-be06-64c0e5c5a5f9"
    "https://dcdn.example/cit/86633e6e-d2da-4185-a285-b3d087a5d711"
  ]
}
```

#### 6.3.3. Individual Trigger Resources

The uCDN can also examine individual triggers:

REQUEST:

```
GET /cit/327df5b8-1df8-4cff-92f8-fda27774c171 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 545
Expires: Sun, 27 Oct 2024 08:49:22 GMT
Server: example-server/0.1
ETag: "554385204989405469"
Last-Modified: Sun, 27 Oct 2024 08:48:17 GMT
Cache-Control: max-age=60
Date: Sun, 27 Oct 2024 08:48:22 GMT
```

```
Content-Type: application/cdni; ptype=ci-trigger.v2

{
  "ctime": 1730119691,
  "etime": 1730119751,
  "mtime": 1730119691,
  "state": "pending",
  "action": "invalidate",
  "specs": [
    {
      "trigger-subject": "metadata",
      "cit-spec-type": "uri-pattern-match",
      "cit-spec-value": {
        "pattern": "https://metadata.example.com/a/b/*"
      }
    },
    {
      "trigger-subject": "content",
      "cit-spec-type": "urls",
      "cit-spec-value": {
        "urls": [ "https://www.example.com/a/index.html" ]
      }
    },
    {
      "trigger-subject": "content",
      "cit-spec-type": "uri-pattern-match",
      "cit-spec-value": {
        "pattern": "https://www.example.com/a/b/*",
        "case-sensitive": true
      }
    }
  ]
}
```

#### 6.3.4. Polling for Changes in Status

The uCDN SHOULD use the ETags and/or Last-Modified headers when polling for changes in trigger collections or the status of individual triggers, as shown in the following examples:

```
   REQUEST:

   GET /cit/state/pending HTTP/1.1
   User-Agent: example-user-agent/0.1
   Host: dcdn.example
   Accept: */*
   If-None-Match: "4331492443626270781"
   If-Modified-Since: Sun, 27 Oct 2024 08:40:23 GMT

   RESPONSE:
   HTTP/1.1 304 Not Modified
   Content-Length: 0
   Expires: Sun, 27 Oct 2024 08:49:21 GMT
   Server: example-server/0.1
   ETag: "4331492443626270781"
   Last-Modified: Sun, 27 Oct 2024 08:48:17 GMT
   Cache-Control: max-age=60
   Date: Sun, 27 Oct 2024 08:48:23 GMT
   Content-Type: application/cdni; ptype=ci-trigger-collection.v2

   REQUEST:

   GET /cit/327df5b8-1df8-4cff-92f8-fda27774c171 HTTP/1.1
   User-Agent: example-user-agent/0.1
   Host: dcdn.example
   Accept: */*
   If-None-Match: "6990548174277557683"
   If-Modified-Since: Sun, 27 Oct 2024 08:49:10 GMT

   RESPONSE:

   HTTP/1.1 304 Not Modified
   Content-Length: 0
   Expires: Sun, 27 Oct 2024 08:49:24 GMT
   Server: example-server/0.1
   ETag: "554385204989405469"
   Last-Modified: Sun, 27 Oct 2024 08:48:17 GMT
   Cache-Control: max-age=60
   Date: Sun, 27 Oct 2024 08:48:24 GMT
   Content-Type: application/cdni; ptype=ci-trigger.v2
```

When the trigger processing is complete, the contents of the filtered collections will be updated. The dCDN SHOULD also update the "ETag" and/or "Last-Modified" response headers - whichever was previously sent - when delivering the updated collection representations. The dCDN SHOULD also use cache control headers, such as "Expires" and "Cache-Control", to indicate how caching of the resource representation should happen by the uCDN and intermediate proxies. For example, when the two example triggers are complete, the collections of pending and complete triggers look as follows:

REQUEST:

```
GET /cit/state/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 51
Expires: Sun, 27 Oct 2024 08:49:25 GMT
Server: example-server/0.1
ETag: "1337503181677633762"
Last-Modified: Sun, 27 Oct 2024 08:48:17 GMT
Cache-Control: max-age=60
Date: Sun, 27 Oct 2024 08:48:25 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection.v2

{
  "staleresourcetime": 86400,
  "triggers": []
}
```

REQUEST:

```
GET /cit/state/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 193
Expires: Sun, 27 Oct 2024 08:49:26 GMT
Server: example-server/0.1
ETag: "4481489539378529796"
```

```
   Last-Modified: Sun, 27 Oct 2024 08:48:17 GMT
   Cache-Control: max-age=60
   Date: Sun, 27 Oct 2024 08:48:26 GMT
   Content-Type: application/cdni; ptype=ci-trigger-collection.v2

   {
     "staleresourcetime": 86400,
     "triggers": [
       "https://dcdn.example/cit/327df5b8-1df8-4cff-92f8-fda27774c171"
     ]
   }
```

### 6.4. Extensions

#### 6.4.1. Execution Policy Extension

This subsection illustrates the uses of the Execution Policy extension. The uCDN can create a dependency between triggers. For example, a preposition trigger should only be processed by the dCDN after a previous purge trigger has been completed.

```
 REQUEST:

 POST /cit HTTP/1.1
 User-Agent: example-user-agent/0.1
 Host: dcdn.example
 Accept: */*
 Content-Type: application/cdni; ptype=ci-trigger.v2
 Content-Length: 294

 {
   "action": "purge",
   "specs": [{
     "trigger-subject": "content",
     "cit-spec-type": "content-objectlist",
     "cit-spec-value": {
       "objects": [{
         "href": "https://www.example.com/hls/1a910c8e/index.m3u8",
         "type": "hls"
       }]
     }
   }]
 }

 RESPONSE:

 HTTP/1.1 201 Created
 Date: Sun, 27 Oct 2024 08:48:27 GMT
```

```
Content-Length: 385
Content-Type: application/cdni; ptype=ci-trigger.v2
Location: https://dcdn.example/cit/564cc45e-9099-4a37-b95e-60342f2647ba
Server: example-server/0.1

{
  "ctime": 1730119707,
  "etime": 1730119767,
  "mtime": 1730119707,
  "state": "pending",
  "action": "purge",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/1a910c8e/index.m3u8",
        "type": "hls"
      }]
     }
  }]
}
```

REQUEST:

```
POST /cit HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger.v2
Content-Length: 527

{
  "action": "preposition",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/09000b67/index.m3u8",
        "type": "hls"
      }]
     }
  }],
  "extensions": [
    "cit-extension-type": "execution-policy",
    "cit-extension-value": {
```

```
      "depends": [
        "https://dcdn.example/cit/564cc45e-9099-4a37-b95e-60342f2647ba"
      ]
    }
  ]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sun, 27 Oct 2024 08:48:28 GMT
Content-Length: 467
Content-Type: application/cdni; ptype=ci-trigger.v2
Location: https://dcdn.example/cit/f6dde35f-703f-49e9-bb80-4964dff3bca5
Server: example-server/0.1

{
  "ctime": 1730119708,
  "etime": 1730119768,
  "mtime": 1730119708,
  "state": "pending",
  "action": "preposition",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/09000b67/index.m3u8",
        "type": "hls"
      }]
     }
  }],
  "extensions": [
    "cit-extension-type": "execution-policy",
    "cit-extension-value": {
      "depends": [
        "https://dcdn.example/cit/564cc45e-9099-4a37-b95e-60342f2647ba"
      ]
    }
  ]
}
```

The uCDN can also stagger long-running triggers to control processing order. In the following example, the uCDN creates a preposition trigger with higher priority, which dCDN should pick up for execution before the earlier triggers.

REQUEST:

```
POST /cit HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger.v2

{
  "action": "preposition",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/b89d49df/index.m3u8",
        "type": "hls"
      }]
     }
  }],
  "extensions": [
    "cit-extension-type": "execution-policy",
    "cit-extension-value": { "priority": 100 }
   ]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sun, 27 Oct 2024 08:48:30 GMT
Content-Length: 526
Content-Type: application/cdni; ptype=ci-trigger.v2
Location: https://dcdn.example/cit/e5483c4a-7c8e-4820-91c8-3c0a9f2edba8
Server: example-server/0.1

{
  "ctime": 1730119710,
  "etime": 1730119770,
  "mtime": 1730119710,
  "state": "pending",
  "action": "preposition",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/b89d49df/index.m3u8",
        "type": "hls"
```

```
      }]
     }
   }],
   "extensions": [
     "cit-extension-type": "execution-policy",
     "cit-extension-value": { "priority": 100 }
    ]
 }
```

#### 6.4.2. Extensions with Error Propagation

In the following example, a CI/T "preposition" command uses two extensions to control the way the trigger is executed. In this example, the receiving dCDN, identified as "AS64500:0", does not support the first extension in the extensions array. The dCDN "AS64500:0" further distributes this trigger to another downstream CDN that is identified as "AS64501:0", which does not support the second extension in the extensions array. The error is propagated from "AS64501:0" to "AS64500:0" and the errors.v2 array reflects both errors.

REQUEST:

```
POST /cit HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger.v2
Content-Length: 1249

{
  "action": "preposition",
  "specs": [{
    "trigger-subject": "content",
    "cit-spec-type": "content-objectlist",
    "cit-spec-value": {
      "objects": [{
        "href": "https://www.example.com/hls/title/index.m3u8",
        "type": "hls"
      }]
    }
  }],
  "extensions": [
    {
      "cit-extension-type": "location-policy",
      "cit-extension-value": {
        "locations": [
          {
```

```
              "action": "allow",
              "footprints": [{
                  "footprint-type": "countrycode",
                  "footprint-value": [ "us" ]
              }]
            },
            {
              "action": "deny",
              "footprints": [{
                 "footprint-type": "countrycode",
                 "footprint-value": [ "ca" ]
              }]
            }
          ]
        },
        "mandatory-to-enforce": true,
        "safe-to-redistribute": true
      },
      {
        "cit-extension-type": "time-policy",
        "cit-extension-value": {
          "unix-time-window": {
            "start": 1730174400,
            "end": 1730260800
          }
        },
        "mandatory-to-enforce": true,
        "safe-to-redistribute": true
      }
    ],
    "cdn-path": [ "AS64496:0" ]
  }

  RESPONSE:

  HTTP/1.1 201 Created
  Date: Sun, 27 Oct 2024 08:48:31 GMT
  Content-Length: 2595
  Content-Type: application/cdni; ptype=ci-trigger.v2
  Location: https://dcdn.example/cit/bcca1cde-ddf0-47db-b859-6a2c043baaa9
  Server: example-server/0.1

  {
    "errors": [
      {
        "extensions": [{
          "cit-extension-type": "location-policy",
          "cit-extension-value": {
```

```
          "locations": [
            {
              "action": "allow",
              "footprints": [{
                  "footprint-type": "countrycode",
                  "footprint-value": [ "us" ]
                }]
            },
            {
              "action": "deny",
              "footprints": [{
                  "footprint-type": "countrycode",
                  "footprint-value": [ "ca" ]
                }]
            }
          ]
        },
        "mandatory-to-enforce": true,
        "safe-to-redistribute": true
      }],
      "description": "unrecognized extension type",
      "error": "eextension",
      "cdn": "AS64500:0"
    },
    {
      "extensions": [{
        "cit-extension-type": "time-policy",
        "cit-extension-value": {
          "unix-time-window": {
            "start": 1730174400,
            "end": 1730260800
          }
        },
        "mandatory-to-enforce": true,
        "safe-to-redistribute": true
      }],
      "description": "unrecognized extension type",
      "error": "eextension",
      "cdn": "AS64501:0"
    }
  ],
  "ctime": 1730119691,
  "etime": 1730119691,
  "mtime": 1730119691,
  "state": "failed",
  "action": "preposition",
  "specs": [{
    "trigger-subject": "content",
```

```
     "cit-spec-type": "content-objectlist",
     "cit-spec-value": {
       "objects": [{
         "href": "https://www.example.com/hls/title/index.m3u8",
         "type": "hls"
       }]
     }
   }],
   "extensions": [
     {
       "cit-extension-type": "location-policy",
       "cit-extension-value": {
         "locations": [
           {
             "action": "allow",
             "footprints": [{
               "footprint-type": "countrycode",
               "footprint-value": [ "us" ]
             }]
           },
           {
             "action": "deny",
             "footprints": [{
               "footprint-type": "countrycode",
               "footprint-value": [ "ca" ]
             }]
           }
         ]
       },
       "mandatory-to-enforce": true,
       "safe-to-redistribute": true
     },
     {
       "cit-extension-type": "time-policy",
       "cit-extension-value": {
         "unix-time-window": {
           "start": 1730174400,
           "end": 1730260800
         }
       },
       "mandatory-to-enforce": true,
       "safe-to-redistribute": true
     }
   ],
   "cdn-path": [ "AS64496:0" ]
 }
```

## 7. IANA Considerations

### 7.1. CDNI Payload Type Parameter Registrations

All references to RFC 8007 in the IANA registries should be replaced with references to this document, apart from references associated with the following registrations:

| Payload Type | Specification |
|---|---|
| ci-trigger-command | RFC 8007 |
| ci-trigger-status | RFC 8007 |
| ci-trigger-collection | RFC 8007 |

Table 12

The IANA is requested to register the following new Payload Types in the "CDNI Payload Types" registry defined by [RFC7736], for use with the "application/cdni" MIME media type.

| Payload Type | Specification |
|---|---|
| ci-trigger.v2 | RFCthis |
| ci-trigger-index.v2 | RFCthis |
| ci-trigger-collection.v2 | RFCthis |
| FCI.CITScope | RFCthis |
| FCI.CITObjectListType | RFCthis |
| FCI.CITEndpoints | RFCthis |
| FCI.CITExtendedStatus | RFCthis |
| FCI.CITPrivateUrlType | RFCthis |

Table 13

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

#### 7.1.1. CDNI ci-trigger.v2 Payload Type

Purpose: The purpose of this payload type is to define a new CI/T trigger object (and any associated capability advertisement)

Interface: CI/T

Encoding: see Section 4.1

#### 7.1.2. CDNI ci-trigger-index.v2 Payload Type

Purpose: The purpose of this payload type is to define a new CI/T trigger index object (and any associated capability advertisement)

Interface: CI/T

Encoding: see Section 4.2

#### 7.1.3. CDNI ci-trigger-collection.v2 Payload Type

Purpose: The purpose of this payload type is to define a new CI/T trigger collection object (and any associated capability advertisement)

Interface: CI/T

Encoding: see Section 4.3

#### 7.1.4. CDNI FCI CI/T Payload Types

##### 7.1.4.1. CDNI FCI CI/T Endpoints Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for CI/T Endpoints objects

Interface: FCI

Value: "FCI.CITEndpoints"

Encoding: see Section 5.1

##### 7.1.4.2. CDNI FCI CI/T Trigger Scope Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for CI/T trigger scope

Interface: FCI

Value: "FCI.CITScope"

Encoding: see Section 5.2

#### 7.1.4.3. CDNI FCI CI/T Object List Type Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for CI/T Object List Type objects

Interface: FCI

Value: "FCI.CITObjectListType"

Encoding: see Section 5.3

#### 7.1.4.4. CDNI FCI CI/T Private URL Type Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for CI/T Private URL Type objects

Interface: FCI

Value: "FCI.CITPrivateUrlType"

Encoding: see Section 5.4

#### 7.1.4.5. CDNI FCI CI/T Extended Status Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for CI/T Extended Status objects

Interface: FCI

Value: "FCI.CITExtendedStatus"

Encoding: see Section 5.5

### 7.2. "CDNI CI/T Trigger Types" Registry For Trigger Actions

In [RFC8007] the IANA was requested to create a new "CDNI CI/T Trigger Types" registry under the "Content Delivery Network Interconnection (CDNI) Parameters" registry group.

Additions to the "CDNI CI/T Trigger Types" registry are made via the RFC Required policy as defined in [RFC8126].

In this second edition of the interface, trigger types are referred to as "trigger actions". The "Trigger Types" registry is used for action definitions. Furthermore, this document, and specifically Section 4.1.1, reuses the definition of "trigger types" as defined in [RFC8007] as trigger actions, and provide their specifications, with no modification compared to [RFC8007].

### 7.3. "CDNI CI/T Trigger Specs" Registry

The IANA is requested to create a new "CDNI CI/T Trigger Specs" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry group. The "CDNI CI/T Trigger Specs" namespace defines the valid trigger targets’ spec values in Section 4.1.2, used by the trigger spec object.

Additions to the "CDNI CI/T Trigger Specs" registry are made via the RFC Required policy as defined in [RFC8126].

The initial contents of the "CDNI CI/T Trigger Specs" registry consist of the names and descriptions listed in Section 4.1.2, with this document serving as their specification.

### 7.4. "CDNI CI/T Trigger Subjects" Registry

The IANA is requested to create a new "CDNI CI/T Trigger Subjects" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry group. The "CDNI CI/T Trigger Subjects" namespace defines the valid trigger targets’ subject values in Section 4.1.2.2, used by the trigger spec object.

Additions to the "CDNI CI/T Trigger Subjects" registry are made via the RFC Required policy as defined in [RFC8126].

The initial contents of the "CDNI CI/T Trigger Subjects" registry consist of the names and descriptions listed in Section 4.1.2.2, with this document serving as their specification.

### 7.5. "CDNI CI/T Object List Types" Registry

The IANA is requested to create a new "CDNI CI/T Object List Types" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry group. The "CDNI CI/T Object List Types" namespace defines the valid object list type values in Section 4.4.2.1, used by the Object List object.

Additions to the "CDNI CI/T Object List Types" registry are made via the Specification Required policy as defined in [RFC8126].

The initial contents of the "CDNI CI/T Object List Types" registry consist of the names and descriptions listed in Section 4.4.2.1, with this document serving as their specification.

### 7.6. "CDNI CI/T Trigger Extensions" Registry

The IANA is requested to create a new "CDNI CI/T Trigger Extensions" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry group. The "CDNI CI/T Trigger Extensions" namespace defines the valid trigger targets' extension values in Section 2.8, used by the trigger spec object.

Additions to the "CDNI CI/T Trigger Extensions" registry are made via the Specification Required policy as defined in [RFC8126].

The initial contents of the "CDNI CI/T Trigger Extensions" registry consist of the names and descriptions listed in Section 2.8, with this document serving as their specification.

### 7.7. "CDNI CI/T Error Codes" Registry

In [RFC8007] the IANA was requested to create a new "CDNI CI/T Error Codes" registry under the "Content Delivery Network Interconnection (CDNI) Parameters" registry group.

Additions to the "CDNI CI/T Error Codes" registry are made via the Specification Required policy as defined in [RFC8126]. The Designated Expert will verify that new Error Code registrations do not duplicate existing Error Code definitions (in name or functionality), prevent gratuitous additions to the namespace, and prevent any additions to the namespace that would impair the interoperability of CDNI implementations.

In this second edition of the interface, the definitions of the Error Codes from [RFC8007] are without change. Additionally, the IANA is requested to register three additional error codes, "espec", "esubject", and "eextension", with the specification as defined in Section 4.1.6.2.

### 7.8. "CDNI CI/T URL Types" Registry

The IANA is requested to create a new "CDNI CI/T URL types" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry group. The "CDNI CI/T URL Types" namespace defines the valid URL type values in Section 4.4.1, used by Section 4.1.2.4, Section 4.1.2.6, Section 4.1.2.7, and Section 4.1.2.8.

The initial contents of the "CDNI CI/T Trigger Extensions" registry consist of the names and descriptions listed in Section 4.4.1, with this document serving as their specification.

# 8. Security Considerations

The CI/T interface provides a mechanism to allow the uCDN to generate requests into the dCDN and to inspect its own CI/T requests and their current states. The CI/T interface does not allow access to, or modification of, the uCDN or the dCDN metadata relating to content delivery or to the content itself. It can only control the presence of that metadata in the dCDN and the processing work and network utilization involved in ensuring that presence.

By examining "preposition" requests to the dCDN, and correctly interpreting content and metadata URLs, an attacker could learn the uCDN’s or content owner’s predictions for future content popularity. By examining "invalidate" or "purge" requests, an attacker could learn about changes in the content owner’s catalog.

An attacker or misbehaving uCDN could inject CI/T triggers to generate processing workload in both the dCDN and uCDN. Similarly, a man-in-the-middle attacker could modify valid trigger requests from the uCDN to achieve the same effect. In both cases, that would decrease the dCDN’s caching efficiency by causing it to unnecessarily acquire or reacquire content metadata and/or content.

The dCDN implementation of CI/T MUST restrict the actions of the uCDN to the data corresponding to that uCDN. Failure to do so would allow the uCDNs to detrimentally affect each other’s efficiency by generating unnecessary acquisition or reacquisition load.

An origin that chooses to delegate its delivery to a CDN is trusting that CDN to deliver content on its behalf; the interconnection of CDNs is an extension of that trust to the dCDNs. That trust relationship is a commercial arrangement, outside the scope of the CDNI protocols. So, while a malicious CDN could deliberately generate load on the dCDN using the CI/T interface, the protocol does not otherwise attempt to address malicious behavior between interconnected CDNs.

## 8.1. Authentication, Authorization, Confidentiality, Integrity Protection

A CI/T implementation MUST support Transport Layer Security (TLS) transport for HTTP (HTTPS) as per [RFC9110].

TLS MUST be used by the server side (dCDN) and the client side (uCDN) of the CI/T interface, including the authentication of the remote end, unless alternate methods are used to ensure the security of the information in the CI/T interface requests and responses (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

The use of TLS for transport of the CI/T interface allows the dCDN and the uCDN to authenticate each other using TLS client authentication and TLS server authentication.

Once the dCDN and the uCDN have mutually authenticated each other, TLS allows:

* The dCDN and the uCDN to authorize each other (to ensure that they are receiving trigger requests from, or responding to, an authorized CDN).

* CDNI commands and responses to be transmitted with confidentiality.

* Protection of the integrity of CDNI commands and responses.

When TLS is used, the general TLS usage guidance in [RFC9325] MUST be followed.

The mechanisms for access control are dCDN-specific and are not standardized as part of this CI/T specification.

HTTP requests that attempt to access or operate on CI/T data belonging to another CDN MUST be rejected using, for example, HTTP 403 ("Forbidden") or 404 ("Not Found"). This is intended to prevent unauthorized users from generating unnecessary load in the dCDNs or the uCDNs due to revalidation, reacquisition, or unnecessary acquisition.

When deploying a network of interconnected CDNs, the possible inefficiencies related to the diamond configuration discussed in Section 2.9 should be considered.

### 8.2. Denial of Service

This document does not define a specific mechanism to protect against Denial-of-Service (DoS) attacks on the CI/T interface. However, CI/T endpoints can be protected against DoS attacks through the use of TLS transport and/or via mechanisms outside the scope of the CI/T interface, such as firewalling or the use of Virtual Private Networks (VPNs).

Depending on the implementation, triggered activity may consume significant processing and bandwidth in the dCDN. A malicious or faulty uCDN could use this to generate unnecessary load in the dCDN. The dCDN should consider mechanisms to avoid overload -- for example, by rate-limiting acceptance or processing of triggers, or by performing batch processing.

### 8.3. Privacy

The CI/T protocol does not carry any information about individual end users of a CDN; there are no privacy concerns for end users.

The CI/T protocol does carry information that could be considered commercially sensitive by CDN operators and content owners. The use of mutually authenticated TLS to establish a secure session for the transport of CI/T data, as discussed in Section 8.1, provides confidentiality while the CI/T data is in transit and prevents parties other than the authorized dCDN from gaining access to that data. The dCDN MUST ensure that it only exposes CI/T data related to the uCDN to clients it has authenticated as belonging to that uCDN.

## 9. References

### 9.1. Normative References

[RFC1930] Hawkinson, J. and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)", BCP 6, RFC 1930, DOI 10.17487/RFC1930, March 1996, <https://www.rfc-editor.org/info/rfc1930>.

[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>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

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

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

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

[RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/info/rfc9325>.

[RFC9388] Sopher, N. and S. Mishra, "Content Delivery Network Interconnection (CDNI) Footprint Types: Country Subdivision Code and Footprint Union", RFC 9388, DOI 10.17487/RFC9388, July 2023, <https://www.rfc-editor.org/info/rfc9388>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

## 9.2. Informative References

[ECMA404] ECMA International, "ECMA-404 - The JSON data interchange syntax", Edition 2, December 2017, <https://ecma-international.org/publications-and-standards/standards/ecma-404/>.

[MPEG-DASH] ISO, "Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment format", ISO/IEC 23009-1:2014, Edition 2, May 2014, <https://www.iso.org/standard/65274.html>.

[MSS] Microsoft, "[MS-SSTR]: Smooth Streaming Protocol", Protocol Revision 8.0, September 2017, <https://msdn.microsoft.com/en-us/library/ff469518.aspx>.

[POSIX.1] "The Open Group Base Specifications Issue 7", IEEE Std 1003.1 2018 Edition, 31 January 2018, <http://pubs.opengroup.org/onlinepubs/9699919799/>.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine , 2000.

[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>.

[RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <https://www.rfc-editor.org/info/rfc7337>.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <https://www.rfc-editor.org/info/rfc7736>.

[RFC7975] Niven-Jenkins, B., Ed. and R. van Brandenburg, Ed., "Request Routing Redirection Interface for Content Delivery Network (CDN) Interconnection", RFC 7975, DOI 10.17487/RFC7975, October 2016, <https://www.rfc-editor.org/info/rfc7975>.

[RFC8216] Pantos, R., Ed. and W. May, "HTTP Live Streaming", RFC 8216, DOI 10.17487/RFC8216, August 2017, <https://www.rfc-editor.org/info/rfc8216>.

## Acknowledgments

The authors thank Kevin Ma for his input, and Carsten Bormann for his review and formalization of the JSON data.

## Authors' Addresses

Nir B. Sopher
Qwilt
6, Ha'harash
Hod HaSharon 4524079
Israel
Email: nir@apache.org

Ori Finkelman
Qwilt
6, Ha'harash
Hod HaSharon 4524079
Israel
Email: ori.finkelman.ietf@gmail.com

Sanjay Mishra
Verizon
13100 Columbia Pike
Silver Spring, MD 20904
United States of America
Email: sanjay.mishra@verizon.com

Jay K. Robertson
Qwilt
275 Shoreline Dr Ste 510
Redwood City, CA 94065
United States of America
Email: jayrobertson@acm.org

Alan Arolovitch
Viasat
1295 Beacon Street Unit 249
Brookline, MA 02446
United States of America
Email: alan.arolovitch@gmail.com