Content Delivery Networks Interconnection
Internet-Draft
Intended status: Standards Track
Expires: 20 April 2026

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

17 October 2025

CDNI Metadata Expression Language
draft-ietf-cdni-metadata-expression-language-00

Abstract

This document specifies the syntax and provides usage examples for an expression language to be used within Content Delivery Network Interconnection (CDNI) Metadata Interface (MI) objects. The purpose of this expression language is to enable metadata to be applied conditionally (based on aspects of an HTTP request), and to enable Hypertext Transfer Protocol (HTTP) responses to be generated or altered dynamically.

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 20 April 2026.

Copyright Notice

Table of Contents

1. Introduction

This document specifies the syntax and provides usage examples for a metadata expression language (MEL) to be used within SVTA/CDNI Metadata Interface (MI) objects, such as those defined within CDNI Metadata Model Extensions [SVTA2029] and its subparts. The expression language allows for inspection and manipulation of HTTP request, response, and header objects as defined in [RFC9110]

The CDNI metadata model defined in [RFC8006] allows metadata to be applied conditionally based on matches to an HTTP request hostname and/or path While this is sufficient for many use cases, the need often arises for configuration metadata (such as caching rules) to be applied conditionally based on matching values of HTTP request headers.

Additionally, content delivery networks (CDNs) often need to conditionally modify responses before forwarding them to a client. These modifications typically involve adding, deleting, or modifying HTTP response headers or synthesizing responses. A set of CDNI metadata objects for conditionally applying metadata and modifying responses is defined in the Processing Stages Metadata Specification [I-D.goldstein-processing-stages-metadata], which requires an expression language to enable the specification of match expressions and synthetically generated values.

2. Requirements

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

3. Expression Usage In CDNI Metadata Model

The CDNI Metadata Expression Language (MEL) provides a syntax with a rich set of variables, operators, and built-in functions to facilitate two key use cases within the CDNI metadata model:

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

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

Refer to the Processing Stages Metadata Specification [I-D.goldstein-processing-stages-metadata] for example usage of both match expressions and value expressions within the Processing Stages Model.

Refer to the Cache Control Metadata Specification [I-D.ietf-cdni-cache-control-metadata] for example usage of Value Expressions to dynamically generate cache keys.

4. Expression Syntax BNF

```
<list>                ::= "[" "]" | "[" <expr_list> "]"
<expr_list>           ::= <expr> | <expr_list> "," <expr>
<expr>                ::= <identifier> "=" <expr> | <function_call> |
                          <expr> <operator> <expr> |
                          <prefix_op> <expr> | <literal>

<function_call>       ::= <function_name> "(" ")" |
                          <function_name> "(" <expr_list> ")"

<operator>            ::= <logic_operator> | <math_operator> |
                          <bit_operator> | <string_operator> |
                          <comparison_operator> | <regex_operator> |
                          <glob_operator> | <ip_operator>

<prefix_op>           ::= "+" | "-" | "!" | "~"

<literal>             ::= NUMBER | STRING | "true" | "false" | "nil"

<logic_operator>      ::= "and" | "or"

<math_operator>       ::= "+" | "-" | "*" | "/" | "%"

<bit_operator>        ::= "|" | "&" | "~" | "<<" | ">>"

<string_operator>     ::= "."

<comparison_operator> ::= "==" | "!=" | ">" | "<" | "<=" | ">="

<regex_operator>      ::= "regexmatch" | "~=" | "regexmatchi" |
                          "!regexmatch" | "!regexmatchi"

<glob_operator>       ::= "globmatch" | "*=" | "globmatchi" |
                          "%*=" | "!globmatch" | "!*=" |
                          "!globmatchi" | "!%*="

<ip_operator>         ::= "ipmatch" | "!ipmatch"

<function_name>       ::= "integer" | "real" | "string" | "boolean" |
                          "upper" | "lower" |
                          "match" | "match_replace" | "add_query" |
                          "add_query_multi" | "remove_query" |
                          "remove_query_multi" |
                          "path_element" | "path_elements"

<identifier>          ::= <alpha_char> <identifier_word> |
                          "_" <identifier_word>
```

```
<identifier_word>     ::= <identifier_char> |
                          <identifier_char> <identifier_word>

<identifier_char>     ::= <alpha_numeric> | "." | "-" | "_" | "#"

<alpha_numeric>       ::= <alpha_char> | <digit>

<alpha_char>          ::= "a-z" | "A-Z"

<digit>               ::= "0" | "1" | "2" | "3" | "4" |
                          "5" | "6" | "7" | "8" | "9"
```

Figure 1

5. Core Variables

The expression language supports two namespaces of core variables, each with a distinct prefix:

* req. Prefix for accessing attributes of the HTTP request object

* resp. Prefix for accessing attributes of the HTTP response object

Variable	Meaning	Type
req.h.<name>	Request header <name>	String
req.uri	Request URI (includes query string and fragment identifier, if any)	String
req.uri.path	Request URI path	String
req.uri.pathquery	Request path and query string	String
req.uri.query	Request query string	String
req.uri.query.<key>	Request query string value associated with <key>. If the key is not present in the uri, nil is returned. If the key is present with no value, as in "a=", then an empty string is returned.	String
req.uri.querykv.<key>	Request query string key and	String

	value associated with <key>, returned as a single String exactly as-is from the request url.For example, when used with a uri containing a query string "key=xxx", expression would return the string "key=xxx". When used with a uri containing a query string "key=", expression would return the string "key=".		
req.method	Request HTTP method (GET, POST, etc.)	String	
req.scheme	Request scheme (http or https)	String	
req.clientip	IP address of the client that made the request. Note: IPv6 addresses MUST NOT be enclosed in square brackets [].	String	
req.clientport	Request port number	Unsigned	
resp.h.<name>	Response header <name>	String	
resp.status	Response status code	Unsigned	

Table 1

6. Operators And Keywords

Operator	Type	Result Type	Meaning
==	infix	Boolean	Equality test
!=	infix	Boolean	Inequality test
!	infix	Boolean	Logical NOT operator
>	infix	Boolean	Greater than test

<	infix	Boolean	Less than test
>=	infix	Boolean	Greater than or equal test
<=	infix	Boolean	Less than or equal
*=	infix	Boolean	Glob style match
~=	infix	Boolean	Regular expression match (see [PCRE] for details on PCRE RegEx matching)
ipmatch	infix	Boolean	Match against IP address or CIDR (IPv4 and IPv6). For an example IP address of '10.2.3.4', matching against '10.2.3.0/24' would return true, while matching against '10.2.3.5' would return false.
+	infix	Numeric	Addition
-	infix	Numeric	Subtraction
*	infix	Numeric	Multiplication
/	infix	Numeric	Division
%	infix	Unsigned or Integer	Modulus
.	infix	String	Concatenation. Note: There MUST be at least 1 space before/after the dot operator.
? :	ternary	*	Conditional operator: <e> ? <v1> : <v2>Evaluates <v1> if <e> is true, <v2> otherwise.
( )	grouping		Used to override precedence and used for function calls.

Table 2

Keyword	Meaning
and	Logical AND
or	Logical OR
not	Logical NOT (see also the "!" operator)
nil	No value (distinct from empty value)
true	Boolean constant: true
false	Boolean constant: false

Table 3

7. Built-In Functions

To enable the types of expressions typically used in content delivery scenarios to evaluate and generate or modify HTTP headers, the following set of built-in functions are defined to facilitate format conversions, matching, and query string management. Any dCDN that advertises support for a metadata property that leverages MEL MUST provide implementations of these built-in functions.

7.1. Type Conversions

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

Table 4

The following table summarizes the return values of type conversion functions when presented with edge-case input values:

Function	non-zero numeric input	nil input	non-numeric string input	numeric zero input
integer(e)	integer	0	0	0
real(e)	real	0	0	0
string(e)	string	'nil'	string	'0'
boolean(e)	true	false	true	false

Table 5

7.2. String Conversions

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

Table 6

7.3. Convenience Functions

Function	Action	Args	Returns
match(string Input, string Match)	Regular expression 'Match' is applied to Input and the matching element (if any) is returned. An empty string is returned if there is no match.See	2	string

	[PCRE] for details on PCRE RegEx matching.			
match_replace(string Input, string Match, string Replace)	Regular expression 'Match' is applied to the Input argument and replaced with the Replace argument upon successful match. It returns the updated (replaced) version of Input.	3	string	
add_query(string Input, string q, string v)	Add query string element q with value v to the Input string. If v is nil, just add the query string element q. The query element q and value v MUST conform to the format defined in [RFC3986].	2	string	
add_query_multi(string input, string qvs)	Add all the query value elements from qvs to the input string. For example, if qvs = "k1=v1, k2=v2, k3=v3"), parameters k1, k2, k3 and associated values would be added to input or replace existing values for these keys in the input.If a qvs element only has a key but no value, the existing value of that key will be kept.	2	string	

remove_query(string Input, string q)	Remove all occurrences of query string element q from the Input string.	2	string
remove_query_multi(string input, string qs)	Remove all occurrences of the query string elements referenced in parameter qs from the input string. For example, if qs= "k1, k2, k3", all occurrences of k1, k2, k3 would be removed from input.	2	string
keep_query_multi(string input, string qs)	Remove all query string elements from input except for the elements referenced in parameter qs. For example, if qs = "k1, k2, k3", all query string elements would be removed from input except for k1, k2, k3.	2	string
path_element(string Input, integer n)	Return the path element n from Input. -1 returns the last element.	2	string
path_elements(string Input, integer n, integer m)	Return the path elements from position n to m.	3	string

Table 7

In the functions add_query_multi, remove_query_multi and keep_query_multi, URI reserved characters as listed in [RFC3986] as well as spaces that need to be part of the key or the value SHOULD be percent-encoded as described in [RFC3986] The input fields ks and kvs SHOULD use only ',' as a separator between the keys, and in the kvs input field use only '=' as a separator between key and values, where neither separator is percent-encoded.

8. User-Defined Variables

In addition to the core variable namespaces that address HTTP request and response objects (req. and resp.), MEL supports user-defined variables that can be referenced via the following prefix:

* var. Prefix for accessing user-defined variables set via MI.SetVariable

A user defined variable is available for access in MEL expressions from the moment it is assigned, throughout the duration of the transaction. If a user variable is accessed before it has been set, it returns the value of nil.

8.1. MI.SetVariable

MI.SetVariable is a GenericMetadata object that allows one to set user-defined variables that can be accessed from MEL. Variables set via this mechanism can be accessed in matching and value construction expressions using a var. prefix before the variable name. Note: Variable names MUST be string literals.

Property: variable-name

* Description: The name of the variable.

* Type: String. Alphanumeric with both uppercase and lowercase characters; the property MUST start with a letter.

* Mandatory-to-Specify: Yes

Property: variable-value

* Description: The string representation of the value to set for the user-defined variable.

* Type: String, either the static value or a MEL expression, determined by the value-is-expression property.

* Mandatory-to-Specify: Yes

Property: value-is-expression

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

* Type: Boolean

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

The following example illustrates the setting of a user-defined variable whose value is a MEL expression suffixed by a static literal string:

```
{
  "generic-metadata-type": "MI.SetVariable",
  "generic-metadata-value": {
    "variable-name": "myvar",
    "variable-value": "req.h.host . ' is my host from player'"
    "value-is-expression": true
  }
}
```

Figure 2

The following example illustrates the setting of a user-defined variable whose value is another user-defined variable suffixed by a static literal string:

```
{
  "generic-metadata-type": "MI.SetVariable",
  "generic-metadata-value": {
    "variable-name": "myvar1",
    "variable-value": "req.host . ' is my host from player'"
    "value-is-expression": true
  }
}
{
  "generic-metadata-type": "MI.SetVariable",
  "generic-metadata-value": {
    "variable-name": "myvar2",
    "variable-value": "var.myvar1 . ' and forwarded to Origin'"
    "value-is-expression": true
  }
}
```

Figure 3

9. Error Handling

9.1. Compile-Time Errors

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

Examples of MEL compile-time errors:

* Unknown MEL variable name referenced in an expression

* Unknown MEL operator, keyword, or functions referenced in an expression

* Incorrect number of arguments used in a MEL expression operator or function

* Incorrect type of argument used in a MEL expression operator or function

9.2. Runtime Errors

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

Examples of runtime errors:

* Failure to allocate memory (or other server resources) when evaluating a MEL expression

* Incorrect runtime argument type in a MEL expression, such as, trying to convert a non-numeric string to a number

10. FCI Capabilities

Since implementing the full MEL specification may be complex and onerous, a mechanism is needed for a dCDN to advertise what portions of the MEL standard it supports (if any). This section introduces the FCI.SupportedMELFeatures object, which can be contained within the Capabilities Advertisement object defined in [RFC8008] section 5, or embedded in an FCI.MetadataExtended object as defined in [SVTA2041], the SVTA Configuration Interface Metadata Capabilities Specification.

If FCI.SupportedMELFeatures is provided within an Capabilities Advertisement object and a FCI.MetadataExtended object for a given footprint, the advertisement within FCI.MetadataExtended MUST take precedence.

10.1. FCI.SupportedMELFeatures

The FCI.SupportedMELFeatures object is contained in FCI Capabilities advertisements and allows a dCDN to advertise the portions of the MEL specification that it supports. If a capabilities advertisement does not contain a FCI.SupportedMELFeatures or a FCI.MetadataExtended object with an embedded FCI.SupportedMELFeatures object, the upstream content delivery network (uCDN) MUST assume that the dCDN has support for the full MEL specification.

Property: keywords

* Description: A list of supported keywords.
* Type: Array
* Mandatory-to-Specify: No. If not specified, it is assumed to be empty.
* Values: ["and", "or", "not", "nil", "true", "false"]

Property: operators

* Description: A list of supported operators.
* Type: Array
* Mandatory-to-Specify: No. If not specified, it is assumed to be empty.
* Values: ["==", "!=", "!", ">", "<", ">=", "<=", "*=", "~=", "+", "-", "*", "/", "%", " . ", "()", "?:", "ipmatch"]

Property: variables

* Description: A list of supported variables.

* Type: Array

* Mandatory-to-Specify: No. If not specified, it is assumed to be empty.

* Values: ["req.h.<name>", "req.uri", "req.uri.path", "req.uri.pathquery", "req.uri.query", "req.uri.query.<key>", "req.uri.querykv.<key>", "req.method", "req.scheme", "resp.h.<name>", "resp.status", "req.clientip", "req.clientport", "var.<user-variable>"]

Property: built-in-functions

* Description: A list of supported built-in function names.

* Type: Array

* Mandatory-to-Specify: No. If not specified, it is assumed to be empty

* Values: ["integer", real", "string", "boolean", upper", "lower", "match", "match_replace", "add_query", "remove_query", "path_element", "path_elements", "add_query_multi", "remove_query_multi", "keep_query_multi"]

The following example advertises a subset of MEL support:

```
{
"capabilities": [
    {
      "capability-type": "FCI.SupportedMELFeatures",
      "capability-value": {
        "keywords": ["and", "or", "not", "nil", "true", "false"],
        "operators" : ["==", "!=", "!", ">", "<", ">=",
           "<=", "~=", " . ", "()" ],
        "variables": ["req.h.<name>", "req.uri", "req.uri.path" ],
        "built-in-functions": [ "string", "boolean", "upper",
           "lower", "match", "match_replace", "path_element"]
      }
    }
  ]
}
```

Figure 4

11. Informative Examples

11.1. MI.ComputedCacheKey

The following examples illustrate usage of the Metadata Expression Language to dynamically generate cache keys, as specified by the MI.ComputedCacheKey object in the Cache Control Metadata Specification [I-D.ietf-cdni-cache-control-metadata]

Example setting the cache key to the value of the X-Cache-Key header from the client HTTP request:

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

Figure 5

Example setting the cache key to the request URI, forced to lower-case:

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

Figure 6

11.2. MI.ExpressionMatch

The following example illustrates usage of the Metadata Expression Language to create a match expression, as specified by the Processing Stages Metadata Specification [I-D.goldstein-processing-stages-metadata]

In this example, the expression is true if the user-agent (glob) matches '*Safari*' and the referrer equals 'www.x.com'.

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

Figure 7

11.3. MI.ResponseTransform

The following example illustrates usage of the Metadata Expression Language to alter the headers of an HTTP response, as specified by the Processing Stages Metadata Specification [I-D.goldstein-processing-stages-metadata]

An HTTP response is transformed by adding a dynamically constructed header with a value that uses the expression language to concatenate the values of the user-agent and host header.

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

Figure 8

12. Security Considerations

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

13. IANA Considerations

13.1. CDNI Payload Types

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

Payload Type	Specification
MI.SetVariable	RFCthis
FCI.SupportedMELFeatures	RFCthis

Table 8: CDNI Payload Types

14. Acknowledgements

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

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

* Pankaj Chaudhari - Disney Streaming Services

* Rajeev RK - picoNETS

* Yoav Gressel - Qwilt

* Alfonso Siloniz - Telefonica

* Ben Rosenblum - Vecima

15. Normative References

[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>.

[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>.

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

[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>.

16. Informative References

[I-D.goldstein-processing-stages-metadata] Goldstein, G., Power, W., and A. Warshavsky, "CDNI Processing Stages Metadata", Work in Progress, Internet-Draft, draft-goldstein-processing-stages-metadata-04, 25 August 2025, <https://datatracker.ietf.org/doc/html/draft-goldstein-processing-stages-metadata-04>.

[I-D.ietf-cdni-cache-control-metadata] Power, W. and G. Goldstein, "CDNI Cache Control Metadata", Work in Progress, Internet-Draft, draft-ietf-cdni-cache-control-metadata-04, 1 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cdni-cache-control-metadata-04>.

[PCRE] PCRE, "Perl Compatible Regular Expressions", <https://www.pcre.org/>.

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

[SVTA2029] SVTA, "CDNI Metadata Model Extensions", <https://svta.org/documents/SVTA2029>.

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

Authors' Addresses

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

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

Arnon Warshavsky
Qwilt
Israel
Email: arnon@qwilt.com