Stream: Internet Engineering Task Force (IETF)
RFC: 9741
Category: Standards Track
Published: March 2025
ISSN: 2070-1721
Author: C. Bormann
Universität Bremen TZI

RFC 9741
Concise Data Definition Language (CDDL): Additional Control Operators for the Conversion and Processing of Text

Abstract

The Concise Data Definition Language (CDDL), standardized in RFC 8610, provides "control operators" as its main language extension point. RFCs have added to this extension point in both an application-specific and a more general way.

The present document defines a number of additional generally applicable control operators for text conversion (bytes, integers, printf-style formatting, and JSON) and for an operation on text.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9741.

Copyright Notice

Table of Contents

1. Introduction

The Concise Data Definition Language (CDDL), standardized in [RFC8610], provides "control operators" as its main language extension point (Section 3.8 of [RFC8610]). RFCs have added to this extension point in both an application-specific [RFC9090] and a more general [RFC9165] way.

The present document defines a number of additional generally applicable control operators. In Table 1, the column marked t is for "target type" (left-hand side), and the column marked c is for "controller type" (right-hand side).

Name	t	c	Purpose
`.b64u`, `.b64c`	text	bytes	Base64 representation of byte strings
`.b64u-sloppy`, `.b64c-sloppy`	text	bytes	Sloppy-tolerant variants of the above
`.hex`, `.hexlc`, `.hexuc`	text	bytes	Base16 representation of byte strings
`.b32`, `.h32`	text	bytes	Base32 representation of byte strings
`.b45`	text	bytes	Base45 representation of byte strings
`.base10`	text	int	Text representation of integer numbers
`.printf`	text	array	Printf-formatted text representation of data items
`.json`	text	any	Text representation of JSON values
`.join`	text or bytes	array	Build text or byte string from array of components

Table 1: Summary of New Control Operators in This Document

1.1. Terminology

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

Regular expressions mentioned in the text are as defined in [RFC9485].

This specification uses terminology from [RFC8610]. In particular, with respect to control operators, "target" refers to the left-hand-side operand and "controller" to the right-hand-side operand. "Tool" refers to tools along the lines of that described in Appendix F of [RFC8610]. Note also that the data model underlying CDDL provides for text strings as well as byte strings as two separate types, which are then collectively referred to as "strings".

The term "opinionated" is used in this document to explain that the selection of operators included is somewhat frugal, based on opinions about what the preferred (and likely) usage scenarios will be. Specifically, not including a potential choice doesn't by itself intend to express that the choice is unacceptable; it might still be added in a future registration if these opinions evolve.

2. Text Conversion

2.1. Byte Strings: Base 16 (Hex), Base 32, Base 45, and Base 64

A CDDL model often defines data that are byte strings in essence but need to be transported in various encoded forms, such as base64 or hex. This section defines a number of control operators to model these conversions.

The control operators generally are of a form that could be used like this:

```
signature-for-json = text .b64u signature
signature = bytes .cbor COSE_Sign1
```

The specification of these control operators cannot provide full coverage of the large number of transformations in use; it focuses on [RFC4648] and additionally [RFC9285], as shown in Table 2. For the representations defined in [RFC4648], this specification uses names as inspired by Section 8 of RFC 8949 [STD94]:

Name	Meaning	Reference
.b64u	Base64url, no padding	Section 5 of [RFC4648]
.b64u-sloppy	Base64url, no padding, sloppy	Section 5 of [RFC4648]
.b64c	Base64 classic, padding	Section 4 of [RFC4648]
.b64c-sloppy	Base64 classic, padding, sloppy	Section 4 of [RFC4648]
.b32	Base32, no padding	Section 6 of [RFC4648]
.h32	Base32 with "Extended Hex" alphabet, no padding	Section 7 of [RFC4648]
.hex	Base16 (hex), either case	Section 8 of [RFC4648]
.hexlc	Base16 (hex), lower case	Section 8 of [RFC4648]
.hexuc	Base16 (hex), upper case	Section 8 of [RFC4648]
.b45	Base45	[RFC9285]

Table 2: Control Operators for Text Conversion of Byte Strings

Note that this specification is somewhat opinionated here: It does not provide base64url or base32(hex) encoding with padding or base64 classic without padding. Experience indicates that these combinations only ever occur in error, so the usability of CDDL is increased by not providing them in the first place. Also, adding "c" makes sure that any decision for classic base64 is actively taken.

These control operators are "strict" in their matching, i.e., they only match base encodings that conform to the mandates of their defining documents. Note that this also means that `.b64u` and `.b64c` only match text strings composed of the set of characters defined for each of them, respectively. (This is perhaps worth pointing out explicitly as it contrasts with the "b64" literal prefix that can be used to notate byte strings in CDDL source code, which simply accepts characters from either alphabet. This behavior is different from the matching behavior of the four base64 control operators defined here.)

The additional designation "sloppy" indicates that the text string is not validated for any additional bits being zero, in variance to what is specified in the paragraph that follows Table 1 in Section 4 of [RFC4648]. Note that the present specification is opinionated again in not specifying a sloppy variant of base32 or base32hex, as no legacy use of sloppy base32(hex) was known at the time of writing. Base45 [RFC9285] is known to be suboptimal for use in environments with limited data transparency (such as URLs) but is included because of its close relationship to QR codes and its wide use in health informatics (note that base45 is strongly specified not to allow sloppy forms of encoding).

2.2. Numerals

Name	Meaning	Reference
`.base10`	Base-ten (decimal) integer	---

Table 3: Control Operator for Text Conversion of Integers

The control operator `.base10` allows the modeling of text strings that carry an integer number in decimal form (as a text string with digits in the usual base-ten positional numeral system), such as in the uint64/int64 formats of YANG-JSON [RFC7951].

```
yang-json-sid = text .base10 (0..9223372036854775807)
```

Again, the specification is opinionated by only providing for integer numbers represented without leading zeros, i.e., the decimal integer numerals match the regular expression `0|-?[1-9][0-9]*` (of course, this is further restricted by the control type). See Section 2.3 for more flexibility and for other numeric bases such as octal, hexadecimal, or binary conversions.

Note that this control operator governs text representations of integers and should not be confused with the control operators governing text representations of byte strings (such as `.b64u`). This contrast is somewhat reinforced by spelling out "base" in the name `.base10` as opposed to those of the byte string operators.

2.3. Printf-Style Formatting

Name	Meaning	Reference
`.printf`	Printf-style formatting of data item(s)	---

Table 4: Control Operator for Printf-Style Formatting of Data Item(s)

The control operator `.printf` allows the modeling of text strings that carry various formatted information, as long as the format can be represented in printf-style formatting strings as they are used in the C language (see Section 7.23.6.1 of [C]; note that the "C23" standard includes %b and %B for formatting into binary digits).

The controller (right-hand side) of the `.printf` control is an array of one printf-style format string and zero or more data items that fit the individual conversion specifications in the format string. The construct matches a text string representing the textual output of an equivalent C-language `printf` function call that receives as arguments the format string and the data items following it in the array.

Out of the functionality described for printf formatting in Section 7.23.6.1 of the C language specification [C], length modifiers (paragraph 7) are not used and **MUST NOT** be included in the format string. The "s" conversion specifier (paragraph 8) is used to interpolate a text string in UTF-8 form. The "c" conversion specifier (paragraph 8) represents a single Unicode scalar value as a UTF-8 character. The "p" and "n" conversion specifiers (paragraph 8) are not used and **MUST NOT** be included in the format string.

In the following example, `my_alg_19` matches the text string `"0x0013"`:

```
my_alg_19 = hexlabel<19>
hexlabel<K> = text .printf (["0x%04x", K])
```

The data items in the controller array do not need to be literals, as in the following example:

```
any_alg = hexlabel<1..20>
hexlabel<K> = text .printf (["0x%04x", K])
```

Here, `any_alg` matches the text strings `"0x0013"` or `"0x0001"` but not `"0x1234"`.

2.4. JSON Values

Some applications store complete JSON texts [STD90] into text strings. The JSON value of these can easily be defined in CDDL by using the default JSON-to-CBOR conversion rules provided in Section 6.2 of RFC 8949 [STD94]. This is supported by a control operator similar to `.cbor` as defined in Section 3.8.4 of [RFC8610].

Name	Meaning	Reference
.json	JSON	[STD90]

Table 5: Control Operator for Text Conversion of JSON Values

```
embedded-claims = text .json claims
claims = {iss: text, exp: text}
```

Notes:

- JSON has known interoperability problems [RFC7493]. While Section 4 of [RFC7493] probably is not relevant to this specification, Section 2 of [RFC7493] provides requirements that need to be followed to make use of the generic data model underlying CDDL. Note that the intention of Section 2.2 of [RFC7493] is directly supported by Section 6.2 of RFC 8949 [STD94]. The recommendation to use text strings for representing numbers outside JSON's interoperable range is a requirement on the application data model and therefore needs to be reflected on the right-hand side of the `.json` control operator.
- This control operator provides no way to constrain the use of blank space or other serialization variants in the JSON representation of the data items; restrictions on the serialization to specific variants (e.g., not providing for the addition of any insignificant blank space and prescribing an order in which map entries are serialized) could be defined in future control operators.
- A `.jsonseq` is not provided in this document for JSON text sequences [RFC7464], as no use case for inclusion in CDDL is known at the time of writing; again, future control operators could address this use case.

3. Text Processing

3.1. Join

Often, text strings need to be constructed out of parts that can best be modeled as an array.

Name	Meaning	Reference
.join	Concatenate elements of an array	---

Table 6: Control Operator for Text Generation from Arrays

For example, an IPv4 address in dotted-decimal might be modeled as in Figure 1.

```
legacy-ip-address = text .join legacy-ip-address-elements
legacy-ip-address-elements = [bytetext, ".", bytetext, ".",
                              bytetext, ".", bytetext]
bytetext = text .base10 byte
byte = 0..255
```

Figure 1: Using the .join Operator to Build Dotted-Decimal IPv4 Addresses

The elements of the controller array need to be strings (text or byte strings). The control operator matches a data item if that data item is also a string, built by concatenating the strings in the array. The result of this concatenation is of the same kind of string (text or bytes) as the first element of the array. (If there is no element in the array, the `.join` construct matches either kind of empty string, obviously further constrained by the control operator target.) The concatenation is performed on the sequences of bytes in the strings. If the result of the concatenation is a text string, the resulting sequence of bytes only matches the target data item if that result is a valid text string (i.e., valid UTF-8). Note that in contrast to the algorithm used in Section 3.2.3 of RFC 8949 [STD94], there is no need for all individual byte sequences going into the concatenation to constitute valid text strings.

Note that this control operator is hard to validate in the most general case, as this would require full parser functionality. Simple implementation strategies will use array elements with constant values as guideposts ("markers", such as the `"."` in Figure 1) for isolating the variable elements that need further validation at the CDDL data model level. Therefore, it is recommended to limit the use of `.join` to simple arrangements where the array elements are laid out explicitly and there are no adjacent variable elements without intervening constant values, and where these constant values do not occur within the text described by the variable elements. If more complex parsing functionality is required, the ABNF control operators (see Section 3 of [RFC9165]) may be useful; however, these cannot reach back into CDDL-specified elements like `.join` can.

> Implementation note: A validator implementation can use the marker elements to scan the text and isolate the variable elements. It also can build a parsing regexp from the elements of the controller array, with capture groups for each element, and validate the captures against the elements of the array. (For more about parsing regexps, see Section 6 of [RFC9485]; see also Section 8 of [RFC9485] for security considerations related to regexps.) In the most general case, these implementation strategies can exhibit false negatives, where the implementation cannot find the structure that would be successfully validated using the controller; it is **RECOMMENDED** that implementations provide full coverage at least for the marker-based subset outlined in the previous paragraph.

4. IANA Considerations

IANA has registered the contents of Table 7 into the "CDDL Control Operators" registry of [IANA.cddl]:

Name	Reference
.b64u	RFC 9741
.b64u-sloppy	RFC 9741
.b64c	RFC 9741
.b64c-sloppy	RFC 9741
.b45	RFC 9741
.b32	RFC 9741
.h32	RFC 9741
.hex	RFC 9741
.hexlc	RFC 9741
.hexuc	RFC 9741
.base10	RFC 9741
.printf	RFC 9741
.json	RFC 9741
.join	RFC 9741

Table 7: New Control Operators

5. Security Considerations

The security considerations in Section 5 of [RFC8610] apply. In addition, for the control operators defined in Section 2.1, the security considerations in Section 12 of [RFC4648] apply.

6. References

6.1. Normative References

[C] International Organization for Standardization, "Information technology - Programming languages - C", Fourth Edition, ISO/IEC 9899:2024, October 2024, <https://www.iso.org/standard/82075.html>. Technically equivalent specification text is available at <https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3220.pdf>.

[IANA.cddl] IANA, "Concise Data Definition Language (CDDL)", <https://www.iana.org/assignments/cddl>.

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/info/rfc9165>.

[RFC9285] Fältström, P., Ljunggren, F., and D.W. van Gulik, "The Base45 Data Encoding", RFC 9285, DOI 10.17487/RFC9285, August 2022, <https://www.rfc-editor.org/info/rfc9285>.

[RFC9485] Bormann, C. and T. Bray, "I-Regexp: An Interoperable Regular Expression Format", RFC 9485, DOI 10.17487/RFC9485, October 2023, <https://www.rfc-editor.org/info/rfc9485>.

[STD90] Internet Standard 90, <https://www.rfc-editor.org/info/std90>.
At the time of writing, this STD comprises the following:

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>.

[STD94] Internet Standard 94, <https://www.rfc-editor.org/info/std94>.
At the time of writing, this STD comprises the following:

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

6.2. Informative References

[RFC7464] Williams, N., "JavaScript Object Notation (JSON) Text Sequences", RFC 7464, DOI 10.17487/RFC7464, February 2015, <https://www.rfc-editor.org/info/rfc7464>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <https://www.rfc-editor.org/info/rfc7493>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC9090] Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", RFC 9090, DOI 10.17487/RFC9090, July 2021, <https://www.rfc-editor.org/info/rfc9090>.

List of Figures

List of Tables

Acknowledgements

Henk Birkholz suggested the need for many of the control operators defined here. The author would like to thank Laurence Lundblade and Jeremy O'Donoghue for sharpening some of the mandates, Mikolai Gütschow for improvements to some examples, A.J. Stein for serving as shepherd for this document and for his shepherd review, the IESG and Directorate reviewers (notably Ari Keränen, Darrel Miller, and Éric Vyncke), and Orie Steele for serving as responsible AD and for providing a detailed AD review.

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org