Thing-to-Thing Research Group
Internet-Draft
Intended status: Experimental
Expires: January 9, 2020

K. Hartke
Ericsson

July 8, 2019

The Constrained RESTful Application Language (CoRAL) draft-hartke-t2trg-coral-09

Abstract

The Constrained RESTful Application Language (CoRAL) defines a data model and interaction model as well as two specialized serialization formats for the description of typed connections between resources on the Web ("links"), possible operations on such resources ("forms"), as well as simple resource metadata.

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 January 9, 2020.

Table of Contents

1. Introduction

The Constrained RESTful Application Language (CoRAL) is a language for the description of typed connections between resources on the Web ("links"), possible operations on such resources ("forms"), as well as simple resource metadata.

CoRAL is intended for driving automated software agents that navigate a Web application based on a standardized vocabulary of link relation types and operation types. It is designed to be used in conjunction with a Web transfer protocol such as the Hypertext Transfer Protocol (HTTP) [RFC7230] or the Constrained Application Protocol (CoAP) [RFC7252].

This document defines the CoRAL data and interaction model, as well as two specialized CoRAL serialization formats.

The CoRAL data and interaction model is a superset of the Web Linking model of RFC 8288 [RFC8288]. The data model consists of two primary elements: "links" that describe the relationship between two resources and the type of that relationship, and "forms" that describe a possible operation on a resource and the type of that operation. Additionally, the data model can describe simple resource metadata in a way similar to the Resource Description Framework (RDF) [W3C.REC-rdf11-concepts-20140225]. In contrast to RDF, the focus of CoRAL however is on the interaction with resources, not just the relationships between them. The interaction model derives from HTML

5 [W3C.REC-html52-20171214] and specifies how an automated software agent can navigate between resources by following links and perform operations on resources by submitting forms.

The primary CoRAL serialization format is a compact, binary encoding of links and forms in Concise Binary Object Representation (CBOR) [RFC7049]. It is intended for environments with constraints on power, memory, and processing resources [RFC7228] and shares many similarities with the message format of the Constrained Application Protocol (CoAP) [RFC7252]: For example, it uses numeric identifiers instead of verbose strings for link relation types and operation types, and pre-parses Uniform Resource Identifiers (URIs) [RFC3986] into (what CoAP considers to be) their components, which simplifies URI processing for constrained nodes a lot. As a result, link serializations in CoRAL are often much more compact than equivalent serializations in CoRE Link Format [RFC6690].

The secondary CoRAL serialization format is a lightweight, textual encoding of links and forms that is intended to be easy to read and write for humans. The format is loosely inspired by the syntax of Turtle [W3C.REC-turtle-20140225] and is mainly intended for giving examples.

1.1. Notational Conventions

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.

Terms defined in this document appear in _cursive_ where they are introduced.

2. Data and Interaction Model

The Constrained RESTful Application Language (CoRAL) is designed for building Web-based applications [W3C.REC-webarch-20041215] in which automated software agents navigate between resources by following links and perform operations on resources by submitting forms.

2.1. Browsing Context

Borrowing from HTML 5 [W3C.REC-html52-20171214], each such agent maintains a _browsing context_ in which the representations of Web resources are processed. (In HTML 5, the browsing context typically corresponds to a tab or window in a Web browser.)

At any time, one representation in each browsing context is designated the _active_ representation.

2.2. Documents

A resource representation in one of the CoRAL serialization formats is called a CoRAL _document_. The Internationalized Resource Identifier (IRI) [RFC3987] that was used to retrieve such a document is called the document's _retrieval context_.

A CoRAL document consists of a list of zero or more links, forms, and embedded resource representations, collectively called _elements_. CoRAL serialization formats may define additional types of elements for efficiency or convenience, such as a base for relative IRI references [RFC3987].

2.3. Links

A _link_ describes a relationship between two resources on the Web [RFC8288]. As defined in RFC 8288, it consists of a _link context_, a _link relation type_, and a _link target_. In CoRAL, a link can additionally have a nested list of zero or more elements, which take the place of link target attributes.

A link can be viewed as a statement of the form "{link context} has a {link relation type} resource at {link target}" where the link target may be further described by nested elements.

The link relation type identifies the semantics of a link. In HTML 5 and RFC 8288, link relation types are typically denoted by an IANA-registered name, such as "stylesheet" or "type". In CoRAL, they are denoted by an IRI such as <http://www.iana.org/assignments/relation/stylesheet> or <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>. This allows for the creation of new link relation types without the risk of collisions when from different organizations or domains of knowledge. An IRI also can lead to documentation, schema, and other information about the link relation type. These IRIs are only used as identity tokens, though, and are compared using Simple String Comparison (Section 5.1 of RFC 3987).

The link context and the link target are both either by an IRI or (similarly to RDF) a literal. If the IRI scheme indicates a Web transfer protocol such as HTTP or CoAP, then an agent can dereference the IRI and navigate the browsing context to the referenced resource; this is called _following the link_. A literal directly identifies a value. This can be a Boolean value, an integer, a floating-point number, a date/time value, a byte string, or a text string.

A link can occur as a top-level element in a document or as a nested element within a link. When a link occurs as a top-level element, the link context implicitly is the document's retrieval context. When a link occurs nested within a link, the link context of the inner link is the link target of the outer link.

There are no restrictions on the cardinality of links; there can be multiple links to and from a particular target, and multiple links of the same or different types between a given link context and target. However, the nested data structure constrains the description of a resource graph to a tree: Links between linked resources can only be described by further nesting links.

2.4. Forms

A _form_ provides instructions to an agent for performing an operation on a Web resource. It consists of a _form context_, an _operation type_, a _request method_, and a _submission target_. Additionally, a form may be accompanied by a list of _form fields_.

A form can be viewed as an instruction of the form "To perform an {operation type} operation on {form context}, make a {request method} request to {submission target}" where the request may be further described by form fields.

The operation type identifies the semantics of the operation. Operation types are denoted like link relation types by an IRI.

The form context is the resource on which an operation is ultimately performed. To perform the operation, an agent needs to construct a request with the specified method and the specified submission target as the request IRI. Usually, the submission target is the same resource as the form context, but it may be a different resource. Constructing and sending the request is called _submitting the form_.

Form fields, specified in the next section, can be used to provide more detailed instructions to the agent for constructing the request. For example, form fields can instruct the agent to include a payload or certain headers in the request that must match the specifications of the form fields.

A form can occur as a top-level element in a document or as a nested element within a link. When a form occurs as a top-level element, the form context implicitly is the document's retrieval context. When a form occurs nested within a link, the form context is the link target of the enclosing link.

2.4.1. Form Fields

Form fields provide further instructions to agents for constructing a request.

For example, a form field could identify one or more data items that need to be included in the request payload or reference another resource (such as a schema) that describes the structure of the payload. A form field could also provide other kinds of information, such as acceptable media types for the payload or expected request headers. Form fields may be specific to the protocol used for submitting the form.

A form field is the pair of a _form field type_ and a _form field value_.

The form field type identifies the semantics of the form field. Form field types are denoted like link relation types and operation types by an IRI.

The form field value can be either an IRI, a Boolean value, an integer, a floating-point number, a date/time value, a byte string, or a text string.

2.5. Embedded Representations

When a document contains links to many resources and an agent needs a representation of each link target, it may be inefficient to retrieve each of these representations individually. To alleviate this, documents can directly embed representations of resources.

An _embedded representation_ consists of a sequence of bytes, labeled with _representation metadata_.

An embedded representation may be a full, partial, or inconsistent version of the representation served from the IRI of the resource.

An embedded representation can occur as a top-level element in a document or as a nested element within a link. When it occurs as a top-level element, it provides an alternate representation of the document's retrieval context. When it occurs nested within a link, it provides a representation of link target of the enclosing link.

2.6. Navigation

An agent begins interacting with an application by performing a GET request on an _entry point IRI_. The entry point IRI is the only IRI an agent is expected to know before interacting with an application.

From there, the agent is expected to make all requests by following links and submitting forms provided by the server in responses. The entry point IRI can be obtained by manual configuration or through some discovery process.

If dereferencing the entry point IRI yields a CoRAL document (or any other representation that implements the CoRAL data and interaction model), then the agent makes this document the active representation in the browsing context and proceeds as follows:

1. The first step for the agent is to decide what to do next, i.e., which type of link to follow or form to submit, based on the link relation types and operation types it understands.

2. The agent then finds the link(s) or form(s) with the respective type in the active representation. This may yield one or more candidates, from which the agent will have to select the most appropriate one. The set of candidates may be empty, for example, when a transition is not supported or not allowed.

3. The agent selects one of the candidates based on the metadata associated with each of these. Metadata includes the content type of the target resource representation, the IRI scheme, the request method, and other information that is provided as nested elements in a link or form fields in a form.

   If the selected candidate contains an embedded representation, the agent MAY skip the following steps and immediately proceed with step 8.

4. The agent obtains the _request IRI_ from the link target or submission target. Fragment identifiers are not part of the request IRI and MUST be separated from the rest of the IRI prior to a dereference.

5. The agent constructs a new request with the request IRI. If the agent is following a link, then the request method MUST be GET. If the agent is submitting a form, then the request method MUST be the one specified in the form. The request IRI may need to be converted to a URI (Section 3.1 of RFC 3987) for protocols that do not support IRIs.

   The agent should set HTTP header fields and CoAP request options according to metadata associated with the link or form (e.g., set the HTTP Accept header field or the CoAP Accept option when the media type of the target resource is provided). Depending on the operation type of a form, the agent may also need to include a

request payload that matches the specifications of one or more form fields.

6. The agent sends the request and receives the response.

7. If a fragment identifier was separated from the request IRI, the agent dereferences the fragment identifier within the received representation.

8. The agent _updates the browsing context_ by making the (embedded or received) representation the active representation.

9. Finally, the agent processes the representation according to the semantics of the content type. If the representation is a CoRAL document (or any other representation that implements the CoRAL data and interaction model), this means the agent has the choice of what to do next again -- and the cycle repeats.

2.7. History Traversal

A browsing context MAY entail a _session history_ that lists the resource representations that the agent has processed, is processing, or will process.

An entry in the session history consists of a resource representation and the request IRI that was used to retrieve the representation. New entries are added to the session history as the agent navigates from resource to resource.

An agent can navigate a browsing context by _traversing the session history_ in addition to following links and submitting forms. For example, if an agent received a representation that doesn't contain any further links or forms, it can revert the active representation back to one it has visited earlier.

Traversing the history should take advantage of caches to avoid new requests. An agent MAY reissue a safe request (e.g., a GET request) when it doesn't have a fresh representation in its cache. An agent MUST NOT reissue an unsafe request (e.g., a PUT or POST request) unless it intends to perform that operation again.

3. Binary Format

This section defines the encoding of documents in the CoRAL binary format.

A document in the binary format is a data item in Concise Binary Object Representation (CBOR) [RFC7049]. The structure of this data

item is presented in the Concise Data Definition Language (CDDL) [RFC8610]. The media type is "application/coral+cbor".

The following restrictions are placed on CBOR encoders: Byte strings and text strings MUST be encoded with definite length. Integers and floating-point values MUST be encoded as such (e.g., a floating-point value of 0.0 must not be encoded as the integer 0).

3.1. Data Structure

The data structure of a document in the binary format is made up of four kinds of elements: links, forms, embedded representations, and (as an extension to the CoRAL data model) base directives. Base directives provide a way to encode IRIs with a common base more efficiently.

Elements are processed in the order they appear in the document. Document processors need to maintain an _environment_ while iterating an array of elements. The environment consists of two variables: the _current context_ and the _current base_. Both the current context and the current base are initially set to the document's retrieval context.

3.1.1. Documents

The body of a document in the binary format is encoded as an array of zero or more links, forms, embedded representations, and directives.

```
document = body

body = [*(link / form / representation / directive)]
```

3.1.2. Links

A link is encoded as an array that consists of the unsigned integer 2, followed by the link relation type and the link target, optionally followed by a link body that contains nested elements.

```
link = [2, relation-type, link-target, ?body]
```

The link relation type is encoded as a text string that conforms to the syntax of an IRI [RFC3987].

```
relation-type = text
```

The link target is denoted by an IRI reference or represented by a literal value. An IRI reference MUST be resolved against the current base. The encoding of and resolution process for IRI references in

the binary format is described in RFC XXXX [I-D.hartke-t2trg-ciri]. The link target may be null, which indicates that the link target is an unidentified resource.

```
link-target = ciri / literal

ciri = <Defined in Section X of RFC XXXX>

literal = bool / int / float / time / bytes / text / null
```

The array of elements in the link body, if any, MUST be processed in a fresh environment. Both the current context and the current base in the new environment are initially set to the link target of the enclosing link.

3.1.3. Forms

A form is encoded as an array that consists of the unsigned integer 3, followed by the operation type and the submission target, optionally followed by a list of form fields.

```
form = [3, operation-type, submission-target, ?form-fields]
```

The operation type is defined in the same way as a link relation type (Section 3.1.2).

```
operation-type = text
```

The request method is either implied by the operation type or encoded as a form field. If there are both, the form field takes precedence over the operation type. Either way, the method MUST be defined for the Web transfer protocol identified by the scheme of the submission target.

The submission target is denoted by an IRI reference. This IRI reference MUST be resolved against the current base.

```
submission-target = ciri
```

3.1.3.1. Form Fields

A list of form fields is encoded as an array of zero or more type-value pairs.

```
form-fields = [*(form-field-type, form-field-value)]
```

The list, if any, MUST be processed in a fresh environment. Both the current context and the current base in the new environment are initially set to the submission target of the enclosing form.

A form field type is defined in the same way as a link relation type (Section 3.1.2).

```
form-field-type = text
```

A form field value can be an IRI reference, a Boolean value, an integer, a floating-point number, a date/time value, a byte string, a text string, or null. An IRI reference MUST be resolved against the current base.

```
form-field-value = ciri / literal
```

3.1.4. Embedded Representations

An embedded representation is encoded as an array that consists of the unsigned integer 0, followed by a byte string containing the representation data, optionally followed by representation metadata.

```
representation = [0, bytes, ?representation-metadata]
```

Representation metadata is encoded as an array of zero or more name-value pairs.

```
representation-metadata = [*(metadata-name, metadata-value)]
```

The metadata, if any, MUST be processed in a fresh environment. All variables in the new environment are initially set to a copy of the variables in the current environment.

The metadata name is defined in the same way as a link relation type (Section 3.1.2).

```
metadata-name = text
```

A metadata value can be an IRI reference, a Boolean value, an integer, a floating-point number, a date/time value, a byte string, a text string, or null. An IRI reference MUST be resolved against the current base.

```
metadata-value = ciri / literal
```

3.1.5. Directives

Directives provide the ability to manipulate the environment when processing a list of elements. There is one type of directives available: the Base directive.

```
directive = base-directive
```

3.1.5.1. Base Directives

A Base directive is encoded as an array that consists of the unsigned integer 1, followed by a base.

```
base-directive = [1, base]
```

The base is denoted by an IRI reference. This IRI reference MUST be resolved against the current context (not the current base).

```
base = ciri
```

The directive is processed by resolving the IRI reference against the current context and assigning the result to the current base.

3.2. Dictionaries

The binary format can reference values from a dictionary to reduce representation size and processing cost. Dictionary references can be used in place of link relation types, link targets, operation types, submission targets, form field types, form field values, representation metadata names, and representation metadata values.

3.2.1. Dictionary References

A dictionary reference is encoded as an unsigned integer. Where a dictionary reference cannot be expressed unambiguously, the unsigned integer is tagged with CBOR tag TBD6.

```
relation-type /= uint

link-target /= #6.TBD6(uint)

operation-type /= uint

submission-target /= #6.TBD6(uint)

form-field-type /= uint

form-field-value /= #6.TBD6(uint)
```

```
metadata-name /= uint

metadata-value /= #6.TBD6(uint)
```

3.2.2. Media Type Parameter

The "application/coral+cbor" media type is defined to have a "dictionary" parameter that specifies the dictionary in use. The dictionary is identified by a URI [RFC3986]. For example, a CoRAL document that uses the dictionary identified by the URI <http://example.com/dictionary> can use the following content type:

```
application/coral+cbor;dictionary="http://example.com/dictionary"
```

The URI serves only as an identifier; it does not necessarily have to be dereferencable (or even use a dereferencable URI scheme). It is permissible, though, to use a dereferencable URI and to serve a representation that provides information about the dictionary in a human- or machine-readable way. (The format of such a representation is outside the scope of this document.)

For simplicity, a CoRAL document can reference values only from one dictionary; the value of the "dictionary" parameter MUST be a single URI. If the "dictionary" parameter is absent, the default dictionary specified in Appendix B of this document is assumed.

Once a dictionary has made an assignment, the assignment MUST NOT be changed or removed. A dictionary, however, may contain additional information about an assignment, which may change over time.

In CoAP [RFC7252], media types (including specific values for media type parameters) are encoded as an unsigned integer called "content format". For use with CoAP, each new CoRAL dictionary MUST register a new content format in the IANA CoAP Content-Formats Registry.

4. Textual Format

This section defines the syntax of documents in the CoRAL textual format using two grammars: The lexical grammar defines how Unicode characters are combined to form line terminators, white space, comments, and tokens. The syntactic grammar defines how tokens are combined to form documents. Both grammars are presented in Augmented Backus-Naur Form (ABNF) [RFC5234].

A document in the textual format is a Unicode string in a Unicode encoding form [UNICODE]. The media type for such documents is "text/coral". The "charset" parameter is not used; charset information is transported inside the document in the form of an OPTIONAL Byte Order

Mark (BOM). The use of the UTF-8 encoding scheme [RFC3629], without a BOM, is RECOMMENDED.

4.1. Lexical Structure

The lexical structure of a document in the textual format is made up of four basic elements: line terminators, white space, comments, and tokens. Of these, only tokens are significant in the syntactic grammar. There are five kinds of tokens: identifiers, IRIs, IRI references, literals, and punctuators.

```
token = identifier / iri / iriref / literal / punctuator
```

When several lexical grammar rules match a sequence of characters in a document, the longest match takes priority.

4.1.1. Line Terminators

Line terminators divide text into lines. A line terminator is any Unicode character with Line_Break class BK, CR, LF, or NL. However, any CR character that immediately precedes a LF character is ignored. (This affects only the numbering of lines in error messages.)

4.1.2. White Space

White space is a sequence of one or more white space characters. A white space character is any Unicode character with the White_Space property.

4.1.3. Comments

Comments are sequences of characters that are ignored when parsing text into tokens. Single-line comments begin with the characters "//" and extend to the end of the line. Delimited comments begin with the characters "/*" and end with the characters "*/". Delimited comments can occupy a portion of a line, a single line, or multiple lines.

Comments do not nest. The character sequences "/*" and "*/" have no special meaning within a single-line comment; the character sequences "//" and "/*" have no special meaning within a delimited comment.

4.1.4. Identifiers

An identifier token is a user-defined symbolic name. The rules for identifiers correspond to those recommended by the Unicode Standard Annex #31 [UNICODE-UAX31] using the following profile:

```
identifier = START *CONTINUE *(MEDIAL 1*CONTINUE)

START = <Any character with the XID_Start property>

CONTINUE = <Any character with the XID_Continue property>

MEDIAL = "-" / "." / "~" / %x58A / %xF0B

MEDIAL =/ %x2010 / %x2027 / %x30A0 / %x30FB
```

All identifiers MUST be converted into Unicode Normalization Form C (NFC), as defined by the Unicode Standard Annex #15 [UNICODE-UAX15]. Comparison of identifiers is based on NFC and is case-sensitive (unless otherwise noted).

4.1.5. IRIs and IRI References

IRIs and IRI references are Unicode strings that conform to the syntax defined in RFC 3987 [RFC3987]. An IRI reference can be either an IRI or a relative reference. Both IRIs and IRI references are enclosed in angle brackets ("<" and ">").

```
iri = "<" IRI ">"

iriref = "<" IRI-reference ">"

IRI = <Defined in Section 2.2 of RFC 3987>

IRI-reference = <Defined in Section 2.2 of RFC 3987>
```

4.1.6. Literals

A literal is a textual representation of a value. There are seven types of literals: Boolean, integer, floating-point, date/time, byte string, text string, and null.

```
literal = boolean / integer / float / datetime / bytes / text

literal =/ null
```

4.1.6.1. Boolean Literals

The case-insensitive tokens "true" and "false" denote the Boolean values true and false, respectively.

```
boolean = "true" / "false"
```

4.1.6.2. Integer Literals

Integer literals denote an integer value of unspecified precision. By default, integer literals are expressed in decimal, but they can also be specified in an alternate base using a prefix: Binary literals begin with "0b", octal literals begin with "0o", and hexadecimal literals begin with "0x".

Decimal literals contain the digits "0" through "9". Binary literals contain "0" and "1", octal literals contain "0" through "7", and hexadecimal literals contain "0" through "9" as well as "A" through "F" in upper- or lowercase.

Negative integers are expressed by prepending a minus sign ("-").

```
integer = ["+" / "-"] (decimal / binary / octal / hexadecimal)

decimal = 1*DIGIT

binary = %x30 (%x42 / %x62) 1*BINDIG

octal = %x30 (%x4F / %x6F) 1*OCTDIG

hexadecimal = %x30 (%x58 / %x78) 1*HEXDIG

DIGIT = %x30-39

BINDIG = %x30-31

OCTDIG = %x30-37

HEXDIG = %x30-39 / %x41-46 / %x61-66
```

4.1.6.3. Floating-point Literals

Floating-point literals denote a floating-point number of unspecified precision.

Floating-point literals consist of a sequence of decimal digits followed by a fraction, an exponent, or both. The fraction consists of a decimal point (".") followed by a sequence of decimal digits. The exponent consists of the letter "e" in upper- or lowercase, followed by an optional sign and a sequence of decimal digits that indicate a power of 10 by which the value preceding the "e" is multiplied.

Negative floating-point values are expressed by prepending a minus sign ("-").

```
float = ["+" / "-"] 1*DIGIT [fraction] [exponent]

fraction = "." 1*DIGIT

exponent = (%x45 / %x65) ["+" / "-"] 1*DIGIT
```

A floating-point literal can additionally denote either the special "Not-a-Number" (NaN) value, positive infinity, or negative infinity. The NaN value is produced by the case-insensitive token "NaN". The two infinite values are produced by the case-insensitive tokens "+Infinity" (or simply "Infinity") and "-Infinity".

```
float =/ "NaN"

float =/ ["+" / "-"] "Infinity"
```

4.1.6.4. Date/Time Literals

Date/time literals denote an instant in time.

A date/time literal consists of the prefix "dt" and a sequence of Unicode characters in Internet Date/Time Format [RFC3339], enclosed in single quotes.

```
datetime = %x64.74 SQUOTE date-time SQUOTE

date-time = <Defined in Section 5.6 of RFC 3339>

SQUOTE = %x27
```

4.1.6.5. Byte String Literals

Byte string literals denote an ordered sequence of bytes.

A byte string literal consists of a prefix and zero or more bytes encoded in Base16, Base32, or Base64 [RFC4648], enclosed in single quotes. Byte string literals encoded in Base16 begin with "h" or "b16", byte string literals encoded in Base32 begin with "b32", and byte string literals encoded in Base64 begin with "b64".

```
bytes = base16 / base32 / base64

base16 = (%x68 / %x62.31.36) SQUOTE <Base16 encoded data> SQUOTE

base32 = %x62.33.32 SQUOTE <Base32 encoded data> SQUOTE

base64 = %x62.36.34 SQUOTE <Base64 encoded data> SQUOTE
```

4.1.6.6. Text String Literals

Text string literals denote a Unicode string.

A text string literal consists of zero or more Unicode characters enclosed in double quotes. It can include simple escape sequences (such as \t for the tab character) as well as hexadecimal and Unicode escape sequences.

```
text = DQUOTE *(char / %x5C escape) DQUOTE

char = <Any character except %x22, %x5C, and line terminators>

escape = simple-escape / hexadecimal-escape / unicode-escape

simple-escape = %x30 / %x62 / %x74 / %x6E / %x76

simple-escape =/ %x66 / %x72 / %x22 / %x27 / %x5C

hexadecimal-escape = (%x78 / %x58) 2HEXDIG

unicode-escape = %x75 4HEXDIG / %x55 8HEXDIG

DQUOTE = %x22
```

An escape sequence denotes a single Unicode code point. For hexadecimal and Unicode escape sequences, the code point is expressed by the hexadecimal number following the "\x", "\X", "\u", or "\U" prefix. Simple escape sequences indicate the code points listed in Table 1.

Escape Sequence	Code Point	Character Name
\0	U+0000	Null
\b	U+0008	Backspace
\t	U+0009	Character Tabulation
\n	U+000A	Line Feed
\v	U+000B	Line Tabulation
\f	U+000C	Form Feed
\r	U+000D	Carriage Return
\"	U+0022	Quotation Mark
\'	U+0027	Apostrophe
\\	U+005C	Reverse Solidus

Table 1: Simple Escape Sequences

4.1.6.7. Null Literal

The case-insensitive tokens "null" and "_" denote the intentional absence of any value.

```
null = "null" / "_"
```

4.1.7. Punctuators

Punctuator tokens are used for grouping and separating.

```
punctuator = "#" / ":" / "*" / "[" / "]" / "{" / "}" / "=" / "->"
```

4.2. Syntactic Structure

The syntactic structure of a document in the textual format is made up of four kinds of elements: links, forms, embedded representations, and (as an extension to the CoRAL data model) directives. Directives provide a way to make documents easier to read and write by setting a base for relative IRI references and introducing shorthands for IRIs.

Elements are processed in the order they appear in the document. Document processors need to maintain an _environment_ while iterating a list of elements. The environment consists of three variables: the _current context_, the _current base_, and the _current mapping from identifiers to IRIs_. Both the current context and the current base are initially set to the document's retrieval context. The current mapping from identifiers to IRIs is initially empty.

4.2.1. Documents

The body of a document in the textual format consists of zero or more links, forms, embedded representations, and directives.

```
document = body

body = *(link / form / representation / directive)
```

4.2.2. Links

A link consists of the link relation type, followed by the link target, optionally followed by a link body enclosed in curly brackets ("{" and "}").

```
link = relation-type link-target ["{" body "}"]
```

The link relation type is denoted by either an IRI, a simple name, or a qualified name.

```
relation-type = iri / simple-name / qualified-name
```

A simple name consists of an identifier. It is resolved to an IRI by looking up the empty string in the current mapping from identifiers to IRIs and appending the specified identifier to the result. It is an error if the empty string is not present in the current mapping.

```
simple-name = identifier
```

A qualified name consists of two identifiers separated by a colon (":"). It is resolved to an IRI by looking up the identifier on the left hand side in the current mapping from identifiers to IRIs and appending the identifier on the right hand side to the result. It is an error if the identifier on the left hand side is not present in the current mapping.

```
qualified-name = identifier ":" identifier
```

The link target is denoted by an IRI reference or represented by a value literal. An IRI reference MUST be resolved against the current base. If the link target is null, the link target is an unidentified resource.

```
link-target = iriref / literal
```

The list of elements in the link body, if any, MUST be processed in a fresh environment. Both the current context and current base in this environment are initially set to the link target of the enclosing link. The mapping from identifiers to IRIs is initially set to a copy of the mapping from identifiers to IRIs in the current environment.

4.2.3. Forms

A form consists of the operation type, followed by a "->" token and the submission target, optionally followed by a list of form fields enclosed in square brackets ("[" and "]").

```
form = operation-type "->" submission-target ["[" form-fields "]"]
```

The operation type is defined in the same way as a link relation type (Section 4.2.2).

```
operation-type = iri / simple-name / qualified-name
```

The request method is either implied by the operation type or encoded as a form field. If there are both, the form field takes precedence over the operation type. Either way, the method MUST be defined for

the Web transfer protocol identified by the scheme of the submission target.

The submission target is denoted by an IRI reference. This IRI reference MUST be resolved against the current base.

```
submission-target = iriref
```

4.2.3.1. Form Fields

A list of form fields consists of zero or more type-value pairs.

```
form-fields = *(form-field-type form-field-value)
```

The list, if any, MUST be processed in a fresh environment. Both the current context and the current base in this environment are initially set to the submission target of the enclosing form. The mapping from identifiers to IRIs is initially set to a copy of the mapping from identifiers to IRIs in the current environment.

The form field type is defined in the same way as a link relation type (Section 4.2.2).

```
form-field-type = iri / simple-name / qualified-name
```

The form field value can be an IRI reference, Boolean literal, integer literal, floating-point literal, byte string literal, text string literal, or null. An IRI reference MUST be resolved against the current base.

```
form-field-value = iriref / literal
```

4.2.4. Embedded Representations

An embedded representation consists of a "*" token, followed by the representation data, optionally followed by representation metadata enclosed in square brackets ("[" and "]").

```
representation = "*" bytes ["[" representation-metadata "]"]
```

Representation metadata consists of zero or more name-value pairs.

```
representation-metadata = *(metadata-name metadata-value)
```

The metadata, if any, MUST be processed in a fresh environment. All variables in the new environment are initially set to a copy of the variables in the current environment.

The metadata name is defined in the same way as a link relation type (Section 4.2.2).

```
metadata-name = iri / simple-name / qualified-name
```

The metadata value can be an IRI reference, Boolean literal, integer literal, floating-point literal, byte string literal, text string literal, or null. An IRI reference MUST be resolved against the current base.

```
metadata-value = iriref / literal
```

4.2.5. Directives

Directives provide the ability to manipulate the environment when processing a list of elements. All directives start with a number sign ("#") followed by a directive identifier. Directive identifiers are case-insensitive and constrained to Unicode characters in the Basic Latin block.

The following two types of directives are available: the Base directive and the Using directive.

```
directive = base-directive / using-directive
```

4.2.5.1. Base Directives

A Base directive consists of a number sign ("#"), followed by the case-insensitive identifier "base", followed by a base.

```
base-directive = "#" "base" base
```

The base is denoted by an IRI reference. The IRI reference MUST be resolved against the current context (not the current base).

```
base = iriref
```

The directive is processed by resolving the IRI reference against the current context and assigning the result to the current base.

4.2.5.2. Using Directives

A Using directive consists of a number sign ("#"), followed by the case-insensitive identifier "using", optionally followed by an identifier and an equals sign ("="), finally followed by an IRI. If the identifier is not specified, it is assumed to be the empty string.

```
using-directive = "#" "using" [identifier "="] iri
```

The directive is processed by adding the specified identifier and IRI to the current mapping from identifiers to IRIs. It is an error if the identifier is already present in the mapping.

5. Usage Considerations

This section discusses some considerations in creating CoRAL-based applications and vocabularies.

5.1. Specifying CoRAL-based Applications

CoRAL-based applications naturally implement the Web architecture [W3C.REC-webarch-20041215] and thus are centered around orthogonal specifications for identification, interaction, and representation:

- Resources are identified by IRIs or represented by value literals.
- Interactions are based on the hypermedia interaction model of the Web and the methods provided by the Web transfer protocol. The semantics of possible interactions are identified by link relation types and operation types.
- Representations are CoRAL documents encoded in the binary format defined in Section 3 or the textual format defined in Section 4. Depending on the application, additional representation formats may be used.

5.1.1. Application Interfaces

Specifications for CoRAL-based applications need to list the specific components used in the application interface and their identifiers. This should include the following items:

- IRI schemes that identify the Web transfer protocol(s) used in the application.
- Internet media types that identify the representation format(s) used in the application, including the media type(s) of the CoRAL serialization format(s).
- Link relation types that identify the semantics of links.
- Operation types that identify the semantics of forms. Additionally, for each operation type, the permissible request method(s).

- Form field types that identify the semantics of form fields. Additionally, for each form field type, the permissible form field values.

- Metadata names that identify the semantics of representation metadata. Additionally, for each metadata name, the permissible metadata values.

5.1.2. Resource Names

Resource names -- i.e., URIs [RFC3986] and IRIs [RFC3987] -- are a cornerstone of Web-based applications. They enable the uniform identification of resources and are used every time a client interacts with a server or a resource representation needs to refer to another resource.

URIs and IRIs often include structured application data in the path and query components, such as paths in a filesystem or keys in a database. It is a common practice in many HTTP-based application programming interfaces (APIs) to make this part of the application specification, i.e., to prescribe fixed URI templates that are hard-coded in implementations. There are a number of problems with this practice [RFC7320], though.

In CoRAL-based applications, resource names are therefore not part of the application specification -- they are an implementation detail. The specification of a CoRAL-based application MUST NOT mandate any particular form of resource name structure. BCP 190 [RFC7320] describes the problematic practice of fixed URI structures in more detail and provides some acceptable alternatives.

5.1.3. Implementation Limits

This document places no restrictions on the number of elements in a CoRAL document or the depth of nested elements. Applications using CoRAL (in particular those running in constrained environments) may wish to limit these numbers and specify implementation limits that an application implementation must at least support to be interoperable.

Applications may also mandate the following and other restrictions:

- use of only either the binary format or the text format;

- use of only either HTTP or CoAP as supported Web transfer protocol;

- use of only dictionary references in the binary format for certain vocabulary;

- use of only either content type strings or content format IDs;
- use of IRI references only up to a specific string length;
- use of CBOR in a canonical format (see Section 3.9 of RFC 7049).

5.2. Minting Vocabulary

New link relation types, operation types, form field types, and metadata names can be minted by defining an IRI [RFC3987] that uniquely identifies the item. Although the IRI can point to a resource that contains a definition of the semantics, clients SHOULD NOT automatically access that resource to avoid overburdening its server. The IRI SHOULD be under the control of the person or party defining it, or be delegated to them.

To avoid interoperability problems, it is RECOMMENDED that only IRIs are minted that are normalized according to Section 5.3 of RFC 3987. Non-normalized forms that are best avoided include:

- Uppercase characters in scheme names and domain names
- Percent-encoding of characters where it is not required by the IRI syntax
- Explicitly stated HTTP default port (e.g., <http://example.com/> is preferable over <http://example.com:80/>)
- Completely empty path in HTTP IRIs (e.g., <http://example.com/> is preferable over <http://example.com>)
- Dot segments ("/./" or "/../") in the path component of an IRI
- Lowercase hexadecimal letters within percent-encoding triplets (e.g., "%3F" is preferable over "%3f")
- Punycode-encoding of Internationalized Domain Names in IRIs
- IRIs that are not in Unicode Normalization Form C [UNICODE-UAX15]

IRIs that identify vocabulary do not need to be registered. The inclusion of domain names in IRIs allows for the decentralized creation of new IRIs without the risk of collisions.

However, IRIs can be relatively verbose and impose a high overhead on a representation. This can be a problem in constrained environments [RFC7228]. Therefore, CoRAL alternatively allows the use of unsigned integers to reference CBOR data items from a dictionary, as specified

in Section 3.2. These impose a much smaller overhead but instead need to be assigned by an authority to avoid collisions.

5.3. Expressing Registered Link Relation Types

Link relation types registered in the IANA Link Relations Registry, such as "collection" [RFC6573] or "icon" [W3C.REC-html52-20171214], can be used in CoRAL by appending the registered name to the IRI <http://www.iana.org/assignments/relation/>:

```
#using iana = <http://www.iana.org/assignments/relation/>

iana:collection </items>
iana:icon       </favicon.png>
```

Note that registered link relation types are required to be lowercased, as per Section 3.3 of RFC 8288 [RFC8288].

(The convention of appending the link relation types to the prefix "http://www.iana.org/assignments/relation/" to form IRIs is adopted from Atom [RFC4287]; see also Appendix A.2 of RFC 8288 [RFC8288].)

5.4. Expressing Simple RDF Statements

An RDF statement [W3C.REC-rdf11-concepts-20140225] says that some relationship, indicated by a predicate, holds between two resources. RDF predicates can therefore be good source for vocabulary to provide resource metadata. For example, a CoRAL document could use the FOAF vocabulary [FOAF] to describe the person or software that made it:

```
#using rdf = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
#using foaf = <http://xmlns.com/foaf/0.1/>

foaf:maker null {
   rdf:type        <http://xmlns.com/foaf/0.1/Person>
   foaf:familyName "Hartke"
   foaf:givenName  "Klaus"
   foaf:mbox       <mailto:klaus.hartke@ericsson.com>
}
```

5.5. Expressing Language-Tagged Strings

Text strings that are the target of a link can be associated with a language tag [RFC5646] by nesting a link of type <http://coreapps.org/base#lang> under the link. The target of this nested link MUST be a text string that conforms to the syntax specified in Section 2.1 of RFC 5646:

```
#using <http://coreapps.org/base#>
#using example = <http://example.org/>
#using iana = <http://www.iana.org/assignments/relation/>

iana:terms-of-service </tos> {
   example:title "Nutzungsbedingungen" { lang "de" }
   example:title "Terms of use"        { lang "en" }
}
```

5.6. Embedding CoRAL in CBOR Data

Data items in the CoRAL binary format (Section 3) may be embedded in other CBOR data [RFC7049] data. Specifications using CDDL [RFC8610] SHOULD reference the following CDDL definitions for this purpose:

```
CoRAL-Document = document

CoRAL-Link = link

CoRAL-Form = form
```

For each embedded document, link, and form, the retrieval context, link context, and form context needs to be specified, respectively.

5.7. Submitting CoRAL Documents

By default, a CoRAL document is a representation that captures the current state of a resource. The meaning of a CoRAL document changes when it is submitted in a request. Depending on the request method, the CoRAL document can capture the intended state of a resource (PUT) or be subject to application-specific processing (POST).

5.7.1. PUT Requests

A PUT request with a CoRAL document enclosed in the request payload requests that the state of the target resource be created or replaced with the state described by the CoRAL document. A successful PUT of a CoRAL document generally means that a subsequent GET on that same target resource would result in an equivalent document being sent in a success response.

An origin server SHOULD verify that a submitted CoRAL document is consistent with any constraints the server has for the target resource. When a document is inconsistent with the target resource, the origin server SHOULD either make it consistent (e.g., by removing inconsistent elements) or respond with an appropriate error message containing sufficient information to explain why the document is unsuitable.

The retrieval context of a CoRAL document in a PUT is the request IRI of the request.

5.7.2. POST Requests

A POST request with a CoRAL document enclosed in the request payload requests that the target resource process the CoRAL document according to the resource's own specific semantics.

The retrieval context of a CoRAL document in a POST is the request IRI of the request.

6. Security Considerations

Parsers of CoRAL documents must operate on input that is assumed to be untrusted. This means that parsers MUST fail gracefully in the face of malicious inputs (e.g., inputs not adhering to the data structure). Additionally, parsers MUST be prepared to deal with resource exhaustion (e.g., resulting from the allocation of big data items) or exhaustion of the call stack (stack overflow).

CoRAL documents intentionally do not feature the equivalent of XML entity references as to preclude the whole class of exponential XML entity expansion ("billion laughs") [CAPEC-197] and improper XML external entity [CAPEC-201] attacks.

Implementers of the CoRAL binary format need to consider the security aspects of processing CBOR with the restrictions described in Section 3. Notably, different number representations for the same numeric value are not equivalent in the CoRAL binary format. See Section 8 of RFC 7049 [RFC7049] for security considerations relating to CBOR.

Implementers of the CoRAL textual format need to consider the security aspects of handling Unicode input. See the Unicode Standard Annex #36 [UNICODE-UAX36] for security considerations relating to visual spoofing and misuse of character encodings. See Section 10 of RFC 3629 [RFC3629] for security considerations relating to UTF-8.

CoRAL makes extensive use of IRIs and URIs. See Section 8 of RFC 3987 [RFC3987] for security considerations relating to IRIs. See Section 7 of RFC 3986 [RFC3986] for security considerations relating to URIs.

The security of applications using CoRAL can depend on the proper preparation and comparison of internationalized strings. For example, such strings can be used to make authentication and authorization decisions, and the security of an application could be

compromised if an entity providing a given string is connected to the wrong account or online resource based on different interpretations of the string. See RFC 6943 [RFC6943] for security considerations relating to identifiers in IRIs and other places.

CoRAL is intended to be used in conjunction with a Web transfer protocol like HTTP or CoAP. See Section 9 of RFC 7230 [RFC7230], Section 9 of RFC 7231 [RFC7231], etc., for security considerations relating to HTTP. See Section 11 of RFC 7252 [RFC7252] for security considerations relating to CoAP.

CoRAL does not define any specific mechanisms for protecting the confidentiality and integrity of CoRAL documents. It relies on application layer or transport layer mechanisms for this, such as Transport Layer Security (TLS) [RFC8446].

CoRAL documents and the structure of a web of resources revealed from automatically following links can disclose personal information and other sensitive information. Implementations need to prevent the unintentional disclosure of such information. See Section of 9 of RFC 7231 [RFC7231] for additional considerations.

Applications using CoRAL ought to consider the attack vectors opened by automatically following, trusting, or otherwise using links and forms in CoRAL documents. Notably, a server that is authoritative for the CoRAL representation of a resource may not necessarily be authoritative for nested elements in the document. See Section 5 of RFC 8288 [RFC8288] for related considerations.

Unless an application mitigates this risk by specifying more specific rules, any link or form in a document where the link or form context and the document's retrieval context don't share the same Web origin [RFC6454] MUST be discarded ("same-origin policy").

7. IANA Considerations

7.1. Media Type "application/coral+cbor"

This document registers the media type "application/coral+cbor" according to the procedures of BCP 13 [RFC6838].

Type name:
   application

Subtype name:
   coral+cbor

Required parameters:

N/A

Optional parameters:
dictionary - See Section 3.2 of [I-D.hartke-t2trg-coral].

Encoding considerations:
binary - See Section 3 of [I-D.hartke-t2trg-coral].

Security considerations:
See Section 6 of [I-D.hartke-t2trg-coral].

Interoperability considerations:
N/A

Published specification:
[I-D.hartke-t2trg-coral]

Applications that use this media type:
See Section 1 of [I-D.hartke-t2trg-coral].

Fragment identifier considerations:
As specified for "application/cbor".

Additional information:
Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): .coral.cbor
Macintosh file type code(s): N/A

Person & email address to contact for further information:
See the Author's Address section of [I-D.hartke-t2trg-coral].

Intended usage:
COMMON

Restrictions on usage:
N/A

Author:
See the Author's Address section of [I-D.hartke-t2trg-coral].

Change controller:
IESG

Provisional registration?
No

7.2. Media Type "text/coral"

This document registers the media type "text/coral" according to the procedures of BCP 13 [RFC6838] and guidelines in RFC 6657 [RFC6657].

Type name:
text

Subtype name:
coral

Required parameters:
N/A

Optional parameters:
N/A

Encoding considerations:
binary - See Section 4 of [I-D.hartke-t2trg-coral].

Security considerations:
See Section 6 of [I-D.hartke-t2trg-coral].

Interoperability considerations:
N/A

Published specification:
[I-D.hartke-t2trg-coral]

Applications that use this media type:
See Section 1 of [I-D.hartke-t2trg-coral].

Fragment identifier considerations:
N/A

Additional information:
Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): .coral
Macintosh file type code(s): N/A

Person & email address to contact for further information:
See the Author's Address section of [I-D.hartke-t2trg-coral].

Intended usage:
COMMON

Restrictions on usage:

N/A

Author:
See the Author's Address section of [I-D.hartke-t2trg-coral].

Change controller:
IESG

Provisional registration?
No

7.3. CoAP Content Formats

This document registers CoAP content formats for the content types "application/coral+cbor" and "text/coral" according to the procedures of RFC 7252 [RFC7252].

- Content Type: application/coral+cbor
  Content Coding: identity
  ID: TBD3
  Reference: [I-D.hartke-t2trg-coral]

- Content Type: text/coral
  Content Coding: identity
  ID: TBD4
  Reference: [I-D.hartke-t2trg-coral]

[[NOTE TO RFC EDITOR: Please replace all occurrences of "TBD3" and "TBD4" in this document with the code points assigned by IANA.]]

[[NOTE TO IMPLEMENTERS: Experimental implementations can use content format ID 65087 for "application/coral+cbor" and content format ID 65343 for "text/coral" until IANA has assigned code points.]]

7.4. CBOR Tag

This document registers a CBOR tag for dictionary references according to the procedures of RFC 7049 [RFC7049].

- Tag: TBD6
  Data Item: unsigned integer
  Semantics: Dictionary reference
  Reference: [I-D.hartke-t2trg-coral]

[[NOTE TO RFC EDITOR: Please replace all occurrences of "TBD6" in this document with the code point assigned by IANA.]]

8. References

8.1. Normative References

[I-D.hartke-t2trg-ciri]
Hartke, K., "Constrained Internationalized Resource Identifiers", draft-hartke-t2trg-ciri-03 (work in progress), July 2019.

[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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[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>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/info/rfc6454>.

[RFC6657] Melnikov, A. and J. Reschke, "Update to MIME regarding "charset" Parameter Handling in Textual Media Types", RFC 6657, DOI 10.17487/RFC6657, July 2012, <https://www.rfc-editor.org/info/rfc6657>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <https://www.rfc-editor.org/info/rfc6943>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[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>.

[UNICODE] The Unicode Consortium, "The Unicode Standard", <http://www.unicode.org/versions/latest/>.

Note that this reference is to the latest version of Unicode, rather than to a specific release. It is not expected that future changes in the Unicode specification will have any impact on this document.

[UNICODE-UAX15] The Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", <http://unicode.org/reports/tr15/>.

[UNICODE-UAX31] The Unicode Consortium, "Unicode Standard Annex #31: Unicode Identifier and Pattern Syntax", <http://unicode.org/reports/tr31/>.

[UNICODE-UAX36]
The Unicode Consortium, "Unicode Standard Annex #36: Unicode Security Considerations", <http://unicode.org/reports/tr36/>.

8.2. Informative References

[CAPEC-197]
MITRE, "CAPEC-197: XML Entity Expansion", July 2018, <https://capec.mitre.org/data/definitions/197.html>.

[CAPEC-201]
MITRE, "CAPEC-201: XML Entity Linking", July 2018, <https://capec.mitre.org/data/definitions/201.html>.

[FOAF] Brickley, D. and L. Miller, "FOAF Vocabulary Specification 0.99", January 2014, <http://xmlns.com/foaf/spec/20140114.html>.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <https://www.rfc-editor.org/info/rfc4287>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <https://www.rfc-editor.org/info/rfc5789>.

[RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012, <https://www.rfc-editor.org/info/rfc6573>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[W3C.REC-html52-20171214]
Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, December 2017, <https://www.w3.org/TR/2017/REC-html52-20171214>.

[W3C.REC-rdf-schema-20140225]
Brickley, D. and R. Guha, "RDF Schema 1.1", World Wide Web Consortium Recommendation REC-rdf-schema-20140225, February 2014, <http://www.w3.org/TR/2014/REC-rdf-schema-20140225>.

[W3C.REC-rdf11-concepts-20140225]
Cyganiak, R., Wood, D., and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax", World Wide Web Consortium Recommendation REC-rdf11-concepts-20140225, February 2014, <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225>.

[W3C.REC-turtle-20140225]
Prud'hommeaux, E. and G. Carothers, "RDF 1.1 Turtle", World Wide Web Consortium Recommendation REC-turtle-20140225, February 2014, <http://www.w3.org/TR/2014/REC-turtle-20140225>.

[W3C.REC-webarch-20041215]
Jacobs, I. and N. Walsh, "Architecture of the World Wide Web, Volume One", World Wide Web Consortium Recommendation REC-webarch-20041215, December 2004, <http://www.w3.org/TR/2004/REC-webarch-20041215>.

Appendix A. Core Vocabulary

This section defines the core vocabulary for CoRAL: a set of link relation types, operation types, form field types, and metadata names.

A.1. Base

Link Relation Types:

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
Indicates that the link's context is an instance of the class specified as the link's target, as defined by RDF Schema [W3C.REC-rdf-schema-20140225].

<http://coreapps.org/base#lang>
Indicates that the link target is a language tag [RFC5646] that specifies the language of the link context.

The link target MUST be a text string in the format specified in Section 2.1 of RFC 5646 [RFC5646].

Operation Types:

<http://coreapps.org/base#update>
Indicates that the state of the form's context can be replaced with the state described by a representation submitted to the server.

This operation type defaults to the PUT method [RFC7231] [RFC7252] for both HTTP and CoAP. Typical overrides by a form field include the PATCH method [RFC5789] [RFC8132] for HTTP and CoAP and the iPATCH method [RFC8132] for CoAP.

<http://coreapps.org/base#search>

Indicates that the form's context can be searched by submitting a search query.

This operation type defaults to the POST method [RFC7231] for HTTP and the FETCH method [RFC8132] for CoAP. Typical overrides by a form field include the POST method [RFC7252] for CoAP.

A.2. Collections

Link Relation Types:

<http://www.iana.org/assignments/relation/item>
Indicates that the link's context is a collection and that the link's target is a member of that collection, as defined in Section 2.1 of RFC 6573 [RFC6573].

<http://www.iana.org/assignments/relation/collection>
Indicates that the link's target is a collection and that the link's context is a member of that collection, as defined in Section 2.2 of RFC 6573 [RFC6573].

Operation Types:

<http://coreapps.org/collections#create>
Indicates that the form's context is a collection and that a new item can be created in that collection with the state defined by a representation submitted to the server.

This operation type defaults to the POST method [RFC7231] [RFC7252] for both HTTP and CoAP.

<http://coreapps.org/collections#delete>
Indicates that the form's context is a member of a collection and that the form's context can be removed from that collection.

This operation type defaults to the DELETE method [RFC7231] [RFC7252] for both HTTP and CoAP.

A.3. HTTP

Form Field Types:

<http://coreapps.org/http#method>
Specifies the HTTP method for the request.

The form field value MUST be a text string in the format defined in Section 4.1 of RFC 7231 [RFC7231]. The set of possible values is maintained in the IANA HTTP Method Registry.

A form field of this type MUST NOT occur more than once in a form. If absent, it defaults to the request method implied by the form's operation type.

<http://coreapps.org/http#accept>
Specifies an acceptable HTTP content type for the request payload. There may be multiple form fields of this type. If a form does not include a form field of this type, the server accepts any or no request payload, depending on the operation type.

The form field value MUST be a text string in the format defined in Section 3.1.1.1 of RFC 7231 [RFC7231]. The possible set of media types and their parameters are maintained in the IANA Media Types Registry.

Representation Metadata:

<http://coreapps.org/http#type>
Specifies the HTTP content type of the representation.

The metadata value MUST be specified as a text string in the format defined in Section 3.1.1.1 of RFC 7231 [RFC7231]. The possible set of media types and their parameters are maintained in the IANA Media Types Registry.

Metadata of this type MUST NOT occur more than once for a representation. If absent, its value defaults to content type "application/octet-stream".

A.4. CoAP

Form Field Types:

<http://coreapps.org/coap#method>
Specifies the CoAP method for the request.

The form field value MUST be an integer identifying one of the CoAP request methods maintained in the IANA CoAP Method Codes Registry (e.g., the integer 2 for the POST method).

A form field of this type MUST NOT occur more than once in a form. If absent, it defaults to the request method implied by the form's operation type.

<http://coreapps.org/coap#accept>
Specifies an acceptable CoAP content format for the request payload. There may be multiple form fields of this type. If a form does not include a form field of this type, the server

accepts any or no request payload, depending on the operation type.

The form field value MUST be an integer identifying one of content formats maintained in the IANA CoAP Content-Formats Registry.

Representation Metadata:

<http://coreapps.org/coap#type>
Specifies the CoAP content format of the representation.

The metadata value MUST be an integer identifying one of content formats maintained in the IANA CoAP Content-Formats Registry.

Metadata of this type MUST NOT occur more than once for a representation. If absent, it defaults to content format 42 (i.e., content type "application/octet-stream" without a content coding).

Appendix B. Default Dictionary

This section defines a default dictionary that is assumed when the "application/coral+cbor" media type is used without a "dictionary" parameter.

Key	Value
0	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
1	<http://www.iana.org/assignments/relation/item>
2	<http://www.iana.org/assignments/relation/collection>
3	<http://coreapps.org/collections#create>
4	<http://coreapps.org/base#update>
5	<http://coreapps.org/collections#delete>
6	<http://coreapps.org/base#search>
7	<http://coreapps.org/coap#accept>
8	<http://coreapps.org/coap#type>
9	<http://coreapps.org/base#lang>
10	<http://coreapps.org/coap#method>

Table 2: Default Dictionary

Acknowledgements

Thanks to Christian Amsuess, Carsten Bormann, Jaime Jimenez, Sebastian Kaebisch, Ari Keranen, Michael Koster, Matthias Kovatsch,

Jim Schaad, and Niklas Widell for helpful comments and discussions that have shaped the document.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com