Workgroup: CoRE Working Group
Published: 9 March 2020
Intended Status: Standards Track
Expires: 10 September 2020
Authors: K. Hartke
Ericsson

The Constrained RESTful Application Language (CoRAL)

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"), and simple resource metadata.

Note to Readers

This note is to be removed before publishing as an RFC.

The issues list for this Internet-Draft can be found at <https://github.com/core-wg/coral/labels/coral>.

Companion material for this Internet-Draft can be found at <https://github.com/core-wg/coral>.

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 10 September 2020.

Copyright Notice

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"), and 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 model and interaction model as well as two specialized CoRAL serialization formats.

1.1. Data and Interaction Model

The data model derives from the Web Linking model of [RFC8288] and consists primarily of two 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.

The data model can additionally make simple statements about resources in a way similar to the Resource Description Framework (RDF) [W3C.REC-rdf11-concepts-20140225]. In contrast to RDF, however, the focus of CoRAL is not on the description of a graph of resources, but on the discovery of possible future application states.

The interaction model derives from the processing model of HTML [W3C.REC-html52-20171214] and specifies how an automated software agent can change the application state by navigating between resources following links and performing operations on resources submitting forms.

1.2. Serialization Formats

The primary serialization format is a compact, binary encoding of links and forms in Concise Binary Object Representation (CBOR) [RFC7049bis]. This format is intended for environments with constraints on power, memory, and processing resources [RFC7228] and

shares many similarities with the message format of CoAP: In place of verbose strings, small numeric identifiers are used to encode link relation types and operation types. Uniform Resource Identifiers (URIs) [RFC3986] are pre-parsed into (what CoAP considers to be) their components, which considerably simplifies URI processing for constrained nodes that already have a CoAP implementation. As a result, link serializations in CoRAL are often much more compact and easier to process than equivalent serializations in CoRE Link Format [RFC6690].

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

1.3. 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 (rendered in plain text as the new term surrounded by underscores).

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, the browsing context typically corresponds to a tab or window in a Web browser.)

At any time, one representation in a 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 URI that was used to retrieve such a document is called the document's *retrieval context*. This URI is

also considered the base URI for relative URI references in the document.

A CoRAL document consists of a list of zero or more links and forms, collectively called *elements*. CoRAL serialization formats may define additional types of elements for efficiency or convenience, such as an embedded base URI that takes precedence over the document's base URI.

2.3. Links

A *link* describes a relationship between two resources on the Web. As in [RFC8288], a link in CoRAL has a *link context*, a *link relation type*, and a *link target*. However, a link in CoRAL does not have target attributes. Instead, a link may have a list of zero or more nested elements. These enable both the description of resource metadata and the chaining of links, which is done in [RFC8288] by setting the anchor of one link to the target of another.

> 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.

A link relation type identifies the semantics of a link. In HTML and in [RFC8288], link relation types are typically denoted by an IANA-registered name, such as stylesheet or type. In CoRAL, all link relation types are in contrast denoted by an Internationalized Resource Identifier (IRI) [RFC3987], 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 decentralized creation of new link relation types without the risk of collisions when from different organizations or domains of knowledge. IRIs can also lead to documentation, schema, and other information about a link relation type. In CoRAL documents, these IRIs are only used as identity tokens, though, and are compared with Simple String Comparison as specified in Section 5.3.1 of [RFC3987].

Link contexts and link targets can both be either a URI, a literal value, or an anonymous resource. If the link target is a URI and the URI scheme indicates a Web transfer protocol like HTTP or CoAP, an agent can dereference the URI and navigate the browsing context to its target resource; this is called *following the link*. Literal values are distinct and distinguishable from URIs and directly identify data by means of a literal representation. A literal value can be either a Boolean value, an integer number, a floating-point number, a date/time instant, a byte string, or a text string. An anonymous resource is a resource that is neither identified by a URI nor a literal representation.

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 nested link is the link target of the enclosing 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 nesting nature of the data model constrains the description of resource relations to a tree: Relations 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 resource on the Web. A form has a *form context*, an *operation type*, a *request method*, and a *submission target*. Additionally, a form may be accompanied by a list of zero or more *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.

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

Form contexts and submission targets are both denoted by a URI. The form context is the resource on which the operation is ultimately performed. To perform the operation, an agent needs to construct a request with the specified method as the request method and the specified submission target as the request URI. Usually, the submission target is the same resource as the form context, but may be a different resource. Constructing and sending the request is called *submitting the form*.

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.

2.5. Form Fields

Form fields can be used to provide more detailed instructions to agents for constructing the request when submitting a form. For example, a form field could instruct an agent to include a certain payload or header field in the request. A payload could, for instance, be described by form fields providing acceptable media

types, a reference to schema information, or a number of individual data items that the agents needs to supply. Form fields can be specific to the Web transfer protocol that is used for submitting the form.

A form field is a pair of a *form field type* and a *form field value*. Additionally, a form field may have a list of zero or more nested elements that further describe the form field value.

A 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.

Form field values can be either a URI, a Boolean value, an integer number, a floating-point number, a date/time instant, a byte string, a text string, or null. A null indicates the intentional absence of any form field value.

2.6. Navigation

An agent begins the interaction with an application by performing a GET request on an *entry point URI*. The entry point URI is the only URI that the agent is expected to know beforehand. From then on, the agent is expected to make all requests by following links and submitting forms that are provided in the responses resulting from the requests. The entry point URI could be obtained through some discovery process or manual configuration.

If dereferencing the entry point URI yields a CoRAL document (or any other representation that implements the CoRAL data and interaction model), 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.

   An agent may follow a link without understanding the link relation type, e.g., for the sake of pre-fetching or building a search index. However, an agent **MUST NOT** submit a form without understanding the operation type.

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 can be empty, for example, when an application state transition is not supported or not allowed.

3. The agent selects one of the candidates based on the metadata associated them (in the form of form fields and nested elements) and their order of appearance in the document. Examples for relevant metadata could include the indication of a media type for the target resource representation, the URI scheme of a target resource, or the request method of an operation.

4. The agent obtains the *request URI* from the link target or submission target. Link targets and submission targets can be denoted by relative URI references, which need to be resolved against a base URI to obtain the request URI. Fragment identifiers are not part of the request URI and **MUST** be separated from the rest of the URI prior to the next step.

5. The agent constructs a new request with the request URI. 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 supplied by the form. An IRI may need to be converted to a URI (see Section 3.1 of [RFC3987]) for protocols that do not support IRIs.

   The agent **SHOULD** set HTTP header fields and CoAP request options according to the metadata (e.g., set the HTTP Accept header field or the CoAP Accept option when a media type for the target resource is provided). Depending on the operation type of a form, the agent may also have to include a request payload that matches the specifications of some form fields.

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

7. If a fragment identifier was separated from the request URI, the agent selects the fragment indicated by the fragment identifier within the received representation according to the semantics of its media type.

8. The agent updates the browsing context by making the (selected fragment of the) received representation the active representation.

9. Finally, the agent processes the representation according to the semantics of its media type. If the representation is a CoRAL document (or any other representation that implements the CoRAL data and interaction model), the agent again has the choice of what to do next. Go to step 1.

2.7. History Traversal

A browsing context has a *session history*, which lists the resource representations that the agent has processed, is processing, or will process.

A session history consists of session history entries. The number of session history entries may be limited and dependent on the agent. An agent with severe constraints on memory size might only have enough memory for the most recent entry.

An entry in the session history consists of a resource representation and the representation's retrieval context. New entries are added to the session history as the agent navigates from resource to resource, discarding entries that are no longer used.

An agent can decide to navigate a browsing context (in addition to following links and submitting forms) by *traversing the session history*. For example, when an agent receives a response with a representation that does not contain any further links or forms, it can navigate back to a resource representation it has visited earlier and make that the active representation.

Traversing the history **SHOULD** take advantage of caches to avoid new requests. An agent may reissue a safe request (e.g., a GET) when it does not have a fresh representation in its cache. An agent **MUST NOT** reissue an unsafe request (e.g., a PUT or POST) unless it actually 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 encoded in Concise Binary Object Representation (CBOR) [RFC7049bis]. The encoding **MUST** satisfy the Core Deterministic Encoding Requirements specified in Section 4.2.1 of [RFC7049bis].

The CBOR structure of a document is presented in the Concise Data Definition Language (CDDL) [RFC8610]. All CDDL rules not defined in this document are defined in Appendix D of [RFC8610].

The media type of documents in the binary format is application/coral+cbor.

3.1. Data Structure

The data structure of a document in the binary format is made up of three kinds of elements: links, forms, and (as an extension to the

CoRAL data model) directives. Directives provide a way to encode URI references with a common base more efficiently.

3.1.1. Documents

A document in the binary format is encoded as a CBOR array that contains zero or more elements. An element is either a link, a form, or a directive.

```
document = [*element]


element = link / form / directive
```

The 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*. The current context and the current base are both initially set to the document's retrieval context.

3.1.2. Directives

Directives provide the ability to manipulate the environment while processing elements.

There is a single type of directives available: the Base directive.

```
directive = base-directive
```

It is an error if a document processor encounters any other type of directive.

3.1.2.1. Base Directives

A Base directive is encoded as a CBOR array that contains the unsigned integer 1 and a base URI.

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

The base URI is denoted by a Constrained Resource Identifier (CRI) reference [I-D.ietf-core-href]. The CRI reference **MUST** be resolved against the current context (not the current base).

```
baseURI = CRI-Reference
```

```
CRI-Reference = <Defined in Section XX of RFC XXXX>
```

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

3.1.3. IRIs

IRIs in links and forms are encoded as CRI references.

```
IRI = CRI-Reference
```

A CRI reference is processed by resolving it to an IRI as specified in Section 5.2 of [I-D.ietf-core-href] using the current base.

3.1.4. Links

A link is encoded as a CBOR array that contains the unsigned integer 2, the link relation type, the link target, and, optionally, an array of zero or more nested elements.

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

The link relation type is an IRI.

```
relation-type = IRI
```

The link target is either an IRI, a literal value, or null.

```
link-target = IRI / literal / null
```

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

The nested elements, if any, **MUST** be processed in a fresh environment. The current context is set to the link target of the enclosing link. The current base is initially set to the link target, if the link target is an IRI; otherwise, it is set to the current base of the current environment.

3.1.5. Forms

A form is encoded as a CBOR array that contains the unsigned integer 3, the operation type, the submission target, and, optionally, an array of zero or more form fields.

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

The operation type is an IRI.

```
operation-type = IRI
```

The submission target is an IRI.

```
submission-target = IRI
```

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

The form fields, if any, **MUST** be processed in a fresh environment. The current context is set to an unspecified URI that represents the enclosing form. The current base is initially set to the submission target of the enclosing form.

3.1.6. Form Fields

A form field is encoded as a CBOR sequence that consists of a form field type, a form field value, and, optionally, an array of zero or more nested elements.

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

The form field type is an IRI.

```
form-field-type = IRI
```

The form field value is either an IRI, a literal value, or null.

```
form-field-value = IRI / literal / null
```

The nested elements, if any, **MUST** be processed in a fresh environment. The current context is set to the form field value of the enclosing form field. The current base is initially set to the form field value, if the form field value is an IRI; otherwise, it is set to the current base of the current environment.

3.2. Dictionary Compression

A document in the binary format **MAY** reference values from an external dictionary. This helps 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, and form field 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, as follows:

```
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)
```

A dictionary reference **MUST NOT** refer to a dictionary value that would otherwise not be syntactically allowed in that position. For example, a dictionary reference in the position of a link relation type cannot refer to a Boolean value; it can only refer to an IRI.

3.2.2. Media Type Parameter

The application/coral+cbor media type for documents in the binary format is defined to have a dictionary parameter that specifies the

dictionary in use. The dictionary is identified by a URI. For example, a CoRAL document that uses the dictionary identified by the URI <http://example.com/dictionary> would have 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 machine- or human-readable way. (The representation format and security considerations of such a representation are 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.

The dictionary parameter is **OPTIONAL**. If it 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, media types (including specific values for their parameters, plus an optional content coding) are encoded as an unsigned integer called the "content format" of a representation. For use with CoAP, each new CoRAL dictionary therefore needs to have a new content format registered in the CoAP Content Formats Registry [CORE-PARAMETERS].

3.3. Export Interface

The definition of documents, links, and forms in the CoRAL binary format can be reused in other CBOR-based protocols. Specifications using CDDL should reference the following rules for this purpose:

```
CoRAL-Document = document
CoRAL-Link = link
CoRAL-Form = form
```

For each embedded document, link, and form, the CBOR-based protocol needs to specify the document retrieval context, link context, and form context, respectively.

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 of textual media types [RFC6657] is not used; instead, 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 three kinds of tokens: identifier tokens, literal tokens, and punctuator tokens.

```
token = identifier / IRIref / boolean / integer / float
      / datetime / bytes / text / null / 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 syntax for identifiers corresponds to the Default Identifier Syntax in Unicode Standard Annex #31 [UAX31] with 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 = <Any character from Table XX>
```

Code Point
U+002D HYPHEN-MINUS
U+002E FULL STOP
U+007E TILDE
U+058A ARMENIAN HYPHEN
U+0F0B TIBETAN MARK INTERSYLLABIC TSHEG
U+2010 HYPHEN
U+2027 HYPHENATION POINT
U+30A0 KATAKANA-HIRAGANA DOUBLE HYPHEN
U+30FB KATAKANA MIDDLE DOT

Table 1: Medial Characters

All identifiers **MUST** be converted into Unicode Normalization Form C (NFC) [Unicode]. Comparison of identifiers is based on NFC and case-sensitive (unless otherwise noted).

4.1.5. Literals

A literal token is a textual representation of a value.

4.1.5.1. IRI Reference Literals

IRI reference tokens denote references to resources on the Web.

An IRI reference literal consists of a Unicode string that conforms to the syntax defined in [RFC3987]. An IRI reference is either an

IRI or a relative reference. IRI references are enclosed in angle brackets (< and >).

```
IRIref = "<" IRI-reference ">"


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

4.1.5.2. Boolean Literals

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

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

4.1.5.3. Integer Literals

Integer literal tokens 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
```

4.1.5.4. Floating-point Literals

Floating-point literal tokens 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" / ["+" / "-"] "Infinity"
```

4.1.5.5. Date/Time Literals

Date/time literal tokens 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
```

4.1.5.6. Byte String Literals

Byte string literal tokens 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
```

4.1.5.7. Text String Literals

Text string literal tokens 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 DQUOTE, %x5C, and line terminators>
escape = simple-escape / hexadecimal-escape / unicode-escape
simple-escape = %x30 / %x62 / %x74 / %x6E / %x76
              / %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 2.

Escape Sequence	Code Point
\0	U+0000 NULL
\b	U+0008 BACKSPACE
\t	U+0009 HORIZONTAL TABULATION
\n	U+000A LINE FEED
\v	U+000B VERTICAL TABULATION
\f	U+000C FORM FEED
\r	U+000D CARRIAGE RETURN
\"	U+0022 QUOTATION MARK

Escape Sequence	Code Point
\'	U+0027 APOSTROPHE
\\	U+005C REVERSE SOLIDUS

Table 2: Simple Escape Sequences

4.1.5.8. Null Literal

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

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

4.1.6. 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 three kinds of elements: links, forms, 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.

4.2.1. Documents

A document in the textual format consists of a sequence of zero or more elements. An element is either a link, a form, or a directive.

```
document = *element
```

```
element = link / form / directive
```

The elements are processed in the order they appear in the document. Document processors need to maintain an *environment* while iterating a sequence of elements. The environment consists of three variables: the *current context*, the *current base*, and the *current mapping from identifiers to IRIs*. The current context and the current base are both initially set to the document's retrieval context. The current mapping from identifiers to IRIs is initially empty.

4.2.2. Directives

Directives provide the ability to manipulate the environment while processing elements.

All directives start with a number sign (#) followed by an identifier. The identifier is case-insensitive and restricted 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
```

It is an error if a document processor encounters any other type of directive.

4.2.2.1. Base Directives

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

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

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

```
baseIRI = IRIref
```

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

4.2.2.2. Using Directives

A Using directive consists of a number sign (#), followed by the case-insensitive token 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 "="] IRIref
```

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 or if the IRI is not an IRI but a relative reference.

4.2.3. IRIs

IRIs in links and forms can be either denoted by an IRI reference or looked up in a mapping from identifiers to IRIs. Lookups can be done in three ways: using a simple name, a qualified name, or a predefined name.

```
IRI = IRIref / simple-name / qualified-name / predefined-name
```

All IRI references and names are processed by resolving them to an IRI, as described in the following sub-sections.

4.2.3.1. IRI References

An IRI reference is resolved to an IRI as specified in Section 6.5 of [RFC3987] using the current base as the base URI.

4.2.3.2. Simple Names

A simple name consists of an identifier.

```
simple-name = identifier
```

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

4.2.3.3. Qualified Names

A qualified name consists of two identifiers separated by a colon (:).

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

A qualified name is resolved to an IRI by looking up the identifier given on the left hand side in the current mapping from identifiers to IRIs. The identifier given on the right hand side is appended to

the result. It is an error if the identifier on the left hand side is not present in the mapping.

4.2.3.4. Predefined Names

A predefined name consists of a commercial at sign (@) followed by an identifier. The identifier is case-insensitive and restricted to Unicode characters in the Basic Latin block.

```
predefined-name = "@" identifier
```

A predefined name is resolved to an IRI by looking up the identifier in Table 3. It is an error if the identifier is not present in the table.

Identifier	IRI
direction	<http://coreapps.org/base#direction>
language	<http://coreapps.org/base#language>

Table 3: Predefined Names

4.2.4. Links

A link consists of the link relation type, followed by the link target, optionally followed by a sequence of zero or more nested elements enclosed in curly brackets ({ and }).

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

The link relation type is an IRI.

```
relation-type = IRI
```

The link target is either an IRI, a literal value, or null.

```
link-target = IRI / literal / null
```

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

The nested elements, if any, **MUST** be processed in a fresh environment. The current context is set to the link target of the enclosing link. The current base is initially set to the link target, if the link target is an IRI; otherwise, it is set to the current base of the current environment. The mapping from identifiers to IRIs is initially set to a copy of the mapping from identifiers to IRIs in the current environment.

4.2.5. Forms

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

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

The operation type is an IRI.

```
operation-type = IRI
```

The submission target is an IRI.

```
submission-target = IRI
```

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

The form fields, if any, **MUST** be processed in a fresh environment. The current context is set to an unspecified URI that represents the enclosing form. The current base is 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.

4.2.6. Form Fields

A form field consists of a form field type, followed by a form field value, optionally followed by a sequence of zero or more nested elements enclosed in curly brackets ({ and }).

```
form-field = form-field-type form-field-value ["{" *element "}"]
```

The form field type is an IRI.

```
form-field-type = IRI
```

The form field value is either an IRI, a literal value, or null.

```
form-field-value = IRI / literal / null
```

The nested elements, if any, **MUST** be processed in a fresh environment. The current context is set to the form field value of the enclosing form field. The current base is initially set to the form field value, if the form field value is an IRI; otherwise, it is set to the current base of the current environment. The mapping from identifiers to IRIs is initially set to a copy of the mapping from identifiers to IRIs in the current environment.

5. Document Semantics

5.1. Submitting 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).

5.1.1. 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 and the base URI of a CoRAL document in a PUT are the request URI of the request.

5.1.2. 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 defined by the target resource's processing semantics; it may be an unspecified URI. The base URI of the document is the request URI of the request.

5.2. Returning Documents

In a response, the meaning of a CoRAL document changes depending on the request method and the response status code. For example, a CoRAL document in a successful response to a GET represents the current state of the target resource, whereas a CoRAL document in a successful response to a POST might represent either the processing result or the new resource state. A CoRAL document in an error response represents the error condition, usually describing the error state and what next steps are suggested for resolving it.

5.2.1. Success Responses

Success responses have a response status code that indicates that the client's request was successfully received, understood, and accepted (2xx in HTTP, 2.xx in CoAP). When the representation in a success response does not describe the state of the target resource, it describes result of processing the request. For example, when a request has been fulfilled and has resulted in one or more new resources being created, a CoRAL document in the response can link to and describe the resource(s) created.

The retrieval context and the base URI of a CoRAL document representing the current state of a resource are the request URI of the request.

The retrieval context of a CoRAL document representing a processing result is an unspecified URI that refers to the processing result itself. The base URI of the document is the request URI of the request.

5.2.2. Redirection Responses

Redirection responses have a response status code that indicates that further action needs to be taken by the agent (3xx in HTTP). A redirection response, for example, might indicate that the target resource is available at a different URI or the server offers a

choice of multiple matching resources, each with its own specific URI.

In the latter case, the representation in the response might contain a list of resource metadata and URI references (i.e., links) from which the agent can choose the most preferred one.

The retrieval context of a CoRAL document representing such multiple choices in a redirection response is an unspecified URI that refers to the redirection itself. The base URI of the document is the request URI of the request.

5.2.3. Error Responses

Error response have a response status code that indicates that either the request cannot be fulfilled or the server failed to fulfill an apparently valid request (4xx or 5xx in HTTP, 4.xx or 5.xx in CoAP). A representation in an error response describes the error condition.

The retrieval context of a CoRAL document representing such an error condition is an unspecified URI that refers to the error condition itself. The base URI of the document is the request URI of the request.

6. Usage Considerations

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

6.1. 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 literal values.

* 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.

6.1.1. 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:

- The Web transfer protocols supported.
- The representation formats used, identified by their Internet media types, including the CoRAL serialization formats.
- The link relation types used.
- The operation types used. Additionally, for each operation type, the permissible request methods.
- The form field types used. Additionally, for each form field type, the permissible form field values.

6.1.2. Resource Identifiers

URIs 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 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 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. However, there are a number of problems with this practice [RFC7320bis].

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.

[RFC7320bis] describes the problematic practice of fixed URI structures in more detail and provides some acceptable alternatives.

6.1.3. 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 limit these numbers and define specific implementation limits that an implementation must support at least 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 the supported Web transfer protocol.
- Use of only dictionary references in the binary format for certain vocabulary.
- Use of URI references and CRI references only up to a specific length.

6.2. Minting Vocabulary

New link relation types, operation types, and form field types can be minted by defining an IRI that uniquely identifies the item. Although the IRI may 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 [RFC3987]. 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

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.

6.3. Expressing Registered Link Relation Types

Link relation types registered in the Link Relations Registry [LINK-RELATIONS], 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>
```

The convention of appending the relation type name to the prefix <http://www.iana.org/assignments/relation/> to form IRIs is adopted from the Atom Syndication Format [RFC4287]; see also Appendix A.2 of [RFC8288].

Note that registered relation type names are required to be lowercase ASCII letters (see Section 3.3 of [RFC8288]).

6.4. Expressing Simple RDF Statements

In RDF [W3C.REC-rdf11-concepts-20140225], a statement says that some relationship, indicated by a predicate, holds between two resources. Existing RDF vocabularies can therefore be a good source for link relation types that describe 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>
}
```

6.5. Expressing Natural Language Texts

Text strings can be associated with a Language Tag [RFC5646] and a base text direction (right-to-left or left-to-right) by nesting links of types <http://coreapps.org/base#language> and <http://coreapps.org/base#direction>, respectively:

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

iana:terms-of-service </tos> {
   base:title "Nutzungsbedingungen" {
      @language  "de"
      @direction "ltr"
   }
   base:title "Terms of use" {
      @language  "en-US"
      @direction "ltr"
   }
}
```

The link relation types <http://coreapps.org/base#language> and <http://coreapps.org/base#direction> are defined in Appendix A.

6.6. Embedding Representations in CoRAL

When a document links to many Web resources and an agent needs a representation of each of them, it can be inefficient to retrieve each representations individually. To minimize round-trips, documents can embed representations of resources.

A representation can be embedded in a document by including a link of type <http://coreapps.org/base#representation>:

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

iana:icon </favicon.gif> {
   base:representation
      b64'R0lGODlhAQABAAAAACH5BAEKAAEALAAAAAABAAEAAAIAOw==' {
      http:type "image/gif"
   }
}
```

An embedded representation **SHOULD** have a nested link of type <http://coreapps.org/http#type> or <http://coreapps.org/coap#type> that indicates the content type of the representation.

The link relation types <http://coreapps.org/base#representation>, <http://coreapps.org/http#type>, and <http://coreapps.org/coap#type> are defined in Appendix A.

7. Security Considerations

CoRAL document processors need to be fully prepared for all types of hostile input that may be designed to corrupt, overrun, or achieve control of the agent processing the document. For example, hostile input may be constructed to overrun buffers, allocate very big data structures, or exhaust the stack depth by setting up deeply nested elements. Processors need to have appropriate resource management to mitigate these attacks.

CoRAL serialization formats intentionally do not feature the equivalent of XML entity references so as to preclude the entire class of attacks relating to them, such as exponential XML entity expansion ("billion laughs") [CAPEC-197] and malicious XML entity linking [CAPEC-201].

Implementers of the CoRAL binary format need to consider the security aspects of decoding CBOR. See Section 10 of [RFC7049bis] for security considerations relating to CBOR. In particular, different number encodings for the same numeric value are not equivalent in CoRAL (e.g., a floating-point value of 0.0 is not the same as the integer 0).

Implementers of the CoRAL textual format need to consider the security aspects of handling Unicode input. See Unicode Technical Report #36 [UTR36] for security considerations relating to visual spoofing and misuse of character encodings. See Section 10 of [RFC3629] for security considerations relating to UTF-8. See Unicode Technical Standard #39 [UTS39] for security mechanisms that can be used to detect possible security problems relating to Unicode.

CoRAL makes extensive use of resource identifiers. See Section 7 of [RFC3986] for security considerations relating to URIs. See Section 8 of [RFC3987] for security considerations relating to IRIs. See Section 7 of [I-D.ietf-core-href] for security considerations relating to CRIs.

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 [RFC6943] for security considerations relating to identifiers in IRIs and other strings.

CoRAL is intended to be used in conjunction with a Web transfer protocol like HTTP or CoAP. See Section 9 of [RFC7230], Section 9 of [RFC7231], etc., for security considerations relating to HTTP. See Section 11 of [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 security mechanisms on the application layer or transport layer 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 9 of [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. See Section 5 of [RFC8288] for related considerations.

In particular, when a CoRAL document is the representation of a resource, the server that is authoritative for that resource may not necessarily be authoritative for nested elements in the document. In this case, unless an application defines specific rules, any link or form where the link/form context and the document's retrieval context do not share the same Web Origin [RFC6454] should be discarded ("same-origin policy").

8. IANA Considerations

8.1. Media Type "application/coral+cbor"

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

Type name:
application

Subtype name:
coral+cbor

Required parameters:
N/A

Optional parameters:
dictionary - See Section 3.2 of [I-D.ietf-core-coral].

Encoding considerations:
binary - See Section 3 of [I-D.ietf-core-coral].

Security considerations:
See Section 7 of [I-D.ietf-core-coral].

Interoperability considerations:
N/A

Published specification:
[I-D.ietf-core-coral]

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

Intended usage:
COMMON

Restrictions on usage:
N/A

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

Change controller:
IESG

Provisional registration?
No

8.2. Media Type "text/coral"

This document registers the media type text/coral according to the procedures of [RFC6838] and guidelines of [RFC6657].

Type name:
text

Subtype name:
coral

Required parameters:
N/A

Optional parameters:
N/A

Encoding considerations:
binary - See Section 4 of [I-D.ietf-core-coral].

Security considerations:
See Section 7 of [I-D.ietf-core-coral].

Interoperability considerations:
N/A

Published specification:
[I-D.ietf-core-coral]

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

Intended usage:
COMMON

Restrictions on usage:
N/A

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

Change controller:
IESG

Provisional registration?
No

8.3. CoAP Content Formats

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

* **Content Type:** application/coral+cbor
  Content Coding: identity
  ID: TBD3
  Reference: [I-D.ietf-core-coral]

* **Content Type:** text/coral
  Content Coding: identity
  ID: TBD4
  Reference: [I-D.ietf-core-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 may use content format ID 65087 for application/coral+cbor and content format ID 65343 for text/coral until IANA has assigned code points.]]

8.4. CBOR Tag

This document registers a CBOR tag for dictionary references according to the procedures of [RFC7049bis].

* **Tag:** TBD6
  Data Item: unsigned integer
  Semantics: Dictionary reference
  Reference: [I-D.ietf-core-coral]

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

9. References

9.1. Normative References

[I-D.ietf-core-href] Hartke, K., "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-03, 9 March 2020, <https://tools.ietf.org/html/draft-ietf-core-href-03>.

[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>.

[RFC7049bis] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", Work in Progress, Internet-Draft, draft-ietf-cbor-7049bis-13, 8 March 2020, <https://tools.ietf.org/html/draft-ietf-cbor-7049bis-13>.

[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, Version 12.1.0", ISBN 978-1-936213-25-2, May 2019, <http://www.unicode.org/versions/Unicode12.1.0/>.

9.2. Informative References

[CAPEC-197] MITRE, "CAPEC-197: XML Entity Expansion", 30 September 2019, <https://capec.mitre.org/data/definitions/197.html>.

[CAPEC-201] MITRE, "CAPEC-201: XML Entity Linking", 30 September 2019, <https://capec.mitre.org/data/definitions/201.html>.

[CORE-PARAMETERS] IANA, "Constrained RESTful Environments (CoRE) Parameters", , <http://www.iana.org/assignments/core-parameters>.

[FOAF] Brickley, D. and L. Miller, "FOAF Vocabulary Specification 0.99", 14 January 2014, <http://xmlns.com/foaf/spec/20140114.html>.

[HAL] Kelly, M., "JSON Hypertext Application Language", Work in Progress, Internet-Draft, draft-kelly-json-hal-08, 12 May

2016, <https://tools.ietf.org/html/draft-kelly-json-hal-08>.

[HTTP-METHODS] IANA, "Hypertext Transfer Protocol (HTTP) Method Registry", , <http://www.iana.org/assignments/http-methods>.

[LINK-RELATIONS] IANA, "Link Relations", , <http://www.iana.org/assignments/link-relations>.

[MEDIA-TYPES] IANA, "Media Types", , <http://www.iana.org/assignments/media-types>.

[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>.

[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>.

[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>.

[RFC7320bis] Nottingham, M., "URI Design and Ownership", Work in Progress, Internet-Draft, draft-nottingham-rfc7320bis-03, 5 January 2020, <https://tools.ietf.org/html/draft-nottingham-rfc7320bis-03>.

[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>.

[UAX31] The Unicode Consortium, "Unicode Standard Annex #31: Unicode Identifier and Pattern Syntax", Revision 31, February 2019, <http://www.unicode.org/reports/tr31/tr31-31.html>.

[UTR36] The Unicode Consortium, "Unicode Technical Report #36: Unicode Security Considerations", Revision 15, September 2014, <http://www.unicode.org/reports/tr36/tr36-15.html>.

[UTS39] The Unicode Consortium, "Unicode Technical Standard #39: Unicode Security Mechanisms", Revision 20, May 2019, <http://www.unicode.org/reports/tr39/tr39-20.html>.

[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, 14 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, 25 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, 25 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, 25 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, 15 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, and form field types.

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#title>
Indicates that the link target is a human-readable label (e.g., a menu entry).

The link target **MUST** be a text string. The text string **SHOULD** be annotated with a language and text direction using nested links of type <http://coreapps.org/base#language> and <http://coreapps.org/base#direction>, respectively.

<http://coreapps.org/base#language>
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 [RFC5646].

<http://coreapps.org/base#direction>
Indicates that the link target is a base text direction (right-to-left or left-to-right) that specifies the text directionality of the link context.

The link target **MUST** be either the text string "rtl" or the text string "ltr".

<http://coreapps.org/base#representation>
Indicates that the link target is a representation of the link context.

The link target **MUST** be a byte string.

The representation may be a full, partial, or inconsistent version of the representation served from the URI of the resource.

A link with this link relation type 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.

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 [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 [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 [RFC7231]. The possible set of values is maintained in the HTTP Methods Registry [HTTP-METHODS].

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 [RFC7231]. The possible set of media types and their parameters is maintained in the Media Types Registry [MEDIA-TYPES].

Link Relation Types:

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

The link target **MUST** be a text string in the format defined in Section 3.1.1.1 of [RFC7231]. The possible set of media types and their parameters is maintained in the Media Types Registry [MEDIA-TYPES].

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 a CoAP method (e.g., the integer 2 for the POST method). The possible set of values is maintained in the CoAP Method Codes Registry [CORE-PARAMETERS].

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 a CoAP content format. The possible set of values is maintained in the CoAP Content Formats Registry [CORE-PARAMETERS].

Link Relation Types:

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

The link target **MUST** be an integer identifying a CoAP content format (e.g., the integer 42 for the content type application/octet-stream without a content coding). The possible set of

values is maintained in the CoAP Content Formats Registry [CORE-PARAMETERS].

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#language>
10	<http://coreapps.org/coap#method>
11	<http://coreapps.org/base#direction>
12	"ltr"
13	"rtl"
14	<http://coreapps.org/base#representation>

Table 4: Default Dictionary

Appendix C. Change Log

This section is to be removed before publishing as an RFC.

Changes from -02 to -03:

- Changed the binary format to express relation types, operation types and form field types using [I-D.ietf-core-href] (#2).
- Clarified the current context and current base for nested elements and form fields (#53).
- Minor editorial improvements (#27).

Changes from -01 to -02:

- Added nested elements to form fields.
- Replaced the special construct for embedded representations with links.

*Changed the textual format to allow simple/qualified names wherever IRI references are allowed.

*Introduced predefined names in the textual format (#39).

*Minor editorial improvements and bug fixes (#16 #28 #31 #37 #39).

Changes from -00 to -01:

*Added a section on the semantics of CoRAL documents in responses.

*Minor editorial improvements.

Acknowledgements

CoRAL is heavily inspired by Mike Kelly's JSON Hypertext Application Language [HAL].

The recommendations for minting IRIs have been adopted from RDF 1.1 Concepts and Abstract Syntax [W3C.REC-rdf11-concepts-20140225] to ease the interoperability between RDF predicates and link relation types.

Thanks to Christian Amsüss, Carsten Bormann, Thomas Fossati, Jaime Jiménez, Jim Schaad, Sebastian Käbisch, Ari Keranen, Michael Koster, Matthias Kovatsch and Niklas Widell for helpful comments and discussions that have shaped the document.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
16483 Stockholm
Sweden

Email: klaus.hartke@ericsson.com