Thing-to-Thing Research Group
Internet-Draft
Intended status: Experimental
Expires: September 14, 2017

K. Hartke
Universitaet Bremen TZI

March 13, 2017

The Constrained RESTful Application Language (CoRAL) draft-hartke-t2trg-coral-02

Abstract

The Constrained RESTful Application Language (CoRAL) is a compact, binary representation format for building RESTful, hypermedia-driven applications that run in constrained environments.

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 http://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 September 14, 2017.

Table of Contents

1. Introduction

Constrained RESTful Environments (CoRE) realize the Web architecture [W3C.REC-webarch-20041215] in a suitable form for constrained nodes and networks [RFC7228].

In the Web, hypertext documents contain links and forms that allow a user to navigate between resources and submit information to a server for processing. By annotating these elements with machine-readable link relation types [RFC5988] and form relation types, it is possible to extend this interaction model to machine-to-machine communication.

This document describes the Constrained RESTful Application Language (CoRAL), a compact serialization format for Web links and forms that is based on Concise Binary Object Representation (CBOR) [RFC7049] and that aligns closely with the Constrained Application Protocol (CoAP) [RFC7252].

1.1. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC5988] and [I-D.hartke-core-apps].

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 [RFC2119].

2. Model

CoRAL is designed for building hypermedia-driven Web applications in which software agents navigate between resources by following links and interact with resources by submitting forms.

Each agent maintains a "browsing context", an environment in which resource representations are processed. (In the traditional Web, the browsing context corresponds to a tab or window in a Web browser.) A browsing context has a session history, which lists the resources that the browsing context has visited, is visiting, or will visit. At any time, one resource in a browsing context is designated the "current" resource. Following a link or submitting a form causes the browsing context to navigate to a new resource.

A link indicates a relationship between two resources, the link context and the link target, and affords the navigation between these. The semantics of the relationship are identified by a link relation type, which in CoRAL can be IANA-registered or application-specific. To minimize round-trips, a link in CoRAL can optionally embed a (complete or partial) representation of the link target. Furthermore, a link target can be an anonymous resource in CoRAL; in this case, the link turns into a "literal" which consists only of a link relation type and a representation.

A form similarly indicates a relationship between two resources, the form context and the form target, and affords the interaction with the context through the submission of the form to the target. In many cases, the target of a form is the same resource as the context, but this is not required. The semantics of a form are identified by a form relation type, which again in CoRAL can be IANA-registered or application-specific.

The submission of a form typically requires the agent to construct a payload that is included with the request. For this purpose, a form indicates the acceptable content formats for the payload and can optionally embed a detailed description of the expected data, for example, as a list of form fields. (The syntax for such a description is outside this document's scope.)

The CoRAL interaction model is as follows:

1. The first step for an 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 form relation types it understands.

2. The agent finds the link(s) or form(s) with the given relation type in the current resource. This may yield one or more candidates from which the agent must select the most appropriate one. The set of candidates may be empty if the transition is not allowed, for example, when the agent is unauthorized. The format of links and forms in CoRAL is specified in Section 3.

3. The agent selects one of the candidates based on the metadata associated with the link or form. Metadata can include the content format of the target resource representation, the URI scheme, the request method and other attributes that describe the target. Metadata is encoded in CoRAL as CoAP-style options, which are specified in Section 4.

4. The agent resolves the URI reference in the link or form to its absolute form in order to obtain the "request URI". CoRAL encodes URI references like CoAP as a sequence of options, which significantly simplifies the implementation of URI processors compared to full RFC 3986 support. The process of reference resolution is specified in Section 5.

5. The agent constructs a new request with the request URI. If the agent follows a link, the request method is GET. If the agent submits a form, the request method is indicated by an option. The agent SHOULD set request parameters according to the link/form attributes (e.g., set the CoAP Accept option when the content format of the target resource is indicated). In the case of a form, the agent also needs to construct a request payload that matches the specifications of the form.

6. Finally, the agent sends the request and retrieves the response. The agent processes the enclosed representation, updates the browsing context to the new resource, and again can decide what to do next.

An agent can additionally navigate a browsing context by traversing the browsing context's session history. The session history consists of a flat list of session history entries, which consist of a URI and may have other information associated with them. Session history entries are added to the session history as the agent navigates from resource to resource. An agent can traverse the session history by updating the browsing context to the resource for that entry.

3. Format

CoRAL can be used as a standalone representation format or embedded in representations in other formats. As a standalone format, CoRAL representations have the media type 'application/coral'. When CoRAL is embedded, it is typically embedded in CBOR-based representation formats, but other representation formats can embed CoRAL as well. The CoRAL format is in all cases the same.

The top-level structure of CoRAL is called a CoRAL document. A CoRAL document consists of a sequence of links, forms, literals and bases, which are collectively called elements. All elements consist of a number indicating the element type, a "href type" that indicates how CoRAL-encoded URI references are to be interpreted in reference resolution, a sequence of zero or more options and, optionally, a body.

Link, form and literal elements come in two flavors: a "fat" format that includes all the items listed above, and a "tiny" format. The tiny formats provide a concise way to express elements that match certain patterns, which are specified below. Base elements are always in the "fat" format. They encode a base URI for reference resolution and apply to all subsequent elements until the next base element is encountered.

In the Web, link relation types are identified by strings, such as 'stylesheet', 'terms-of-service' or 'item'. In order to minimize the overhead of using these relation types in constrained environments, [I-D.hartke-core-apps] extends the IANA Link Relation Types registry with a numeric identifier for each type. CoRAL uses these numeric identifiers instead of the textual names. The same optimization is applied to form relation types, CoAP content formats and CoAP request methods.

Applications MAY use negative numbers to indicate application-specific link relation types and form relation types, which do not need to be IANA-registered. The mapping from numbers to textual names needs to be provided by a CoRAL profile, which is indicated by the optional 'profile' parameter of the 'application/coral' media type (see Section 6). The 'application/coral' media type without a

'profile' parameter does not define any application-specific values. A representation format embedding CoRAL documents MAY define application-specific values for the CoRAL documents as well.

CoRAL defines a number of options that can be included in elements. Options are used to encode the relation type, the target resource URI and target attributes. Options are serialized in CBOR as a sequence of unwrapped pairs where each pair consists of a CoRAL option number and an option value. The pairs in a CoRAL element MUST be sorted such that the option numbers appear in ascending order.

Using the notation of [I-D.greevenbosch-appsawg-cbor-cddl], the CoRAL data format can be expressed as follows:

```
document      = [*element]
element       = tiny-link / tiny-literal / tiny-form
              / fat-link  / fat-literal  / fat-form
              / base

tiny-link     = [1, href-type, relation]
tiny-literal  = [2, href-type, relation, format, body]
tiny-form     = [3, href-type, relation, accept]
base          = [4, href-type, options]
fat-link      = [5, href-type, options, ?body]
fat-literal   = [6, href-type, options, body]
fat-form      = [7, href-type, options, ?body]

href-type     = &(append-relation: 0,
                  absolute-path: 1,
                  append-path: 2,
                  relative-path: 3)

relation      = int
format        = uint
accept        = uint
options       = [*(option-number, option-value)]
option-number = uint
option-value  = uint / int / text / bytes
body          = bytes
```

The tiny formats expand as follows:

```
          [1, H, R]         ->   [5, H, [1, R]]
          [2, H, R, F, B]   ->   [6, H, [1, R, 4, F], B]
          [3, H, R, A]      ->   [7, H, [1, R, 3, A]]
```

4. Options

Table 1 summarizes the CoRAL options defined in this document.

No.	R	Name	Format	Length	Default
1	x	Relation	int		(none)
2		Method	uint		2 (POST)
3	x	Accept	uint		(none)
4	x	Format	uint		(none)
5		Href.Scheme	text	1-255	(none)
6		Href.Host.Name	text	1-255	(none)
7		Href.Host.IPv4	bytes	4	(none)
8		Href.Host.IPv6	bytes	16	(none)
9		Href.Port	uint		(see below)
10	x	Href.Path	text	0-255	(none)
11	x	Href.Query	text	0-255	(none)
12		Href.Fragment	text	0-255	(none)
13		Title	text	0-255	(none)
14		Updatable	bool		false
15		Deletable	bool		false

Table 1: Options

The option properties are defined as follows:

Number: An option is identified by an option number.

Repeatable (R): An option that is repeatable MAY be included one or more times in an element. An option that is not repeatable MUST NOT be included more than once. If an agent encounters an option with more occurrences than the option is defined for, each extra occurrence MUST be ignored.

Format: Option values are defined to have a certain format, which is the CBOR encoding of the specified type.

Length: Option values with types "text" and "bytes" are defined to have a specific length, often in the form of an upper and lower bound. The length of an option value MUST NOT be outside the defined range. If an agent encounters an option with a length outside the defined range, that option MUST be ignored.

Default Value: Options can be defined to have a default value. If the value of an option is intended to be this default value, the

option SHOULD NOT be included in the element. If the option is not present, the default value MUST be assumed.

The semantics of the individual options are specified in the following sections.

4.1. Accept

The Accept Option indicates the acceptable content formats for the representation included in a form submission. The option value of an Accept Option is one of the content format IDs registered in the CoAP Content-Formats registry. If a form does not include an Accept Option, the service accepts any content format.

4.2. Deletable

The Deletable Option, when present in a link, defines a form that can be used to delete the link target. The form relation type of that form is 'delete', the context and target of the form are identical to the target of the link, the submission method is DELETE and no representation must be submitted.

4.3. Format

The Format Option, when present in a link or a form, provides a hint indicating what the content format of the payload of the CoAP response should be when following the link or submitting the form. Note that this is only a hint; it does not override the Content-Format Option included in the CoAP response. If the Format Option occurs more than once, an agent SHOULD set the Accept Option in the CoAP request message to request a particular content format.

The Format Option is REQUIRED if a link embeds a representation in the link body. The Format Option is also REQUIRED in a literal. In both cases, the first occurrence of the option indicates the content format of the embedded representation; any additional occurrences indicate available alternative content formats.

The option value of a Format Option is one of the content format IDs registered in the CoAP Content-Formats registry.

4.4. Href.*

The Href.Scheme, Href.Host.Name, Href.Host.IPv4, Href.IPv6, Href.Port, Href.Path, Href.Query and Href.Fragment Options are used to specify the target resource URI of a link or form. They hold the following values:

- the Href.Scheme Option specifies the URI scheme name,
- the Href.Host.Name Option specifies the host as a registered name,
- the Href.Host.IPv4 Option specifies the host as a 32-bit IPv4 address,
- the Href.Host.IPv6 Option specifies the host as a 128-bit IPv6 address,
- the Href.Port Option specifies the port number,
- each Href.Path Option specifies one segment of the path,
- each Href.Query Option specifies one argument of the query, and
- the Href.Fragment Option specifies the fragment identifier.

The Href.Host.Name, Href.Host.IPv4 and Href.Host.IPv6 options are mutually exclusive.

The default value of the Href.Port Option is the default port for the URI scheme.

Table 2 lists the permitted Href.* options by Href Type. A 'yes' indicates that an option of this type MAY be present; a 'no' indicates that an option of this type MUST NOT be present. The resolution of a sequence of Href.* options against a base URI is specified in Section 5.

	absolute-path	relative-path	append-path	append-relation
Href.Scheme	yes	no	no	no
Href.Host.Name	yes	no	no	no
Href.Host.IPv4	yes	no	no	no
Href.Host.IPv6	yes	no	no	no
Href.Port	yes	no	no	no
Href.Path	yes	yes	yes	no
Href.Query	yes	yes	yes	no
Href.Fragment	yes	yes	yes	no

Table 2: Permitted Href.* Options by Href Type

4.5. Method

The Method Option, when present in a form, indicates the CoAP method to use for form submission. The option value is one of the CoAP method codes registered in the CoAP Method Codes registry. The option value defaults to the POST method when the Method Option is not present in a form.

4.6. Relation

The Relation Option indicates the link relation type of a link or literal and the form relation type of a form. At least one Relation Option is REQUIRED in each link, literal and form element.

In a link or literal, the option value of a Relation Option is either one of the link relation type IDs registered in the Link Relation Types registry (>= 0) or one of the application-specific link relation type IDs defined by the CoRAL profile (< 0).

In a form, the option value of Relation Option is either one of the form relation type IDs registered in the Form Relation Types registry (>= 0) or one of the application-specific form relation type IDs defined by the CoRAL profile (< 0).

4.7. Title

The Title Option, when present, is used to label the target of a link such that it can be used as a human-readable identifier (e.g., a menu entry).

4.8. Updatable

The Updatable Option, when present in a link, defines a form that can be used to update the link target. The form relation type of that form is 'update', the context and target of the form are identical to the target of the link, the submission method is PUT and the content format of the submitted representation must be one of the formats indicated by the Format Option in the link.

5. Reference Resolution

This section defines the process of resolving a URI reference within a link or form to an absolute URI suitable for inclusion in a CoAP request.

5.1. Establish a Base URI

URI references can be relative and thus are only usable when a base URI is known. This means that a base URI must be established before the use of all URI references that might be relative.

The base URI of a reference in a link or form is established as specified in Section 5.1 of [RFC3986]. CoRAL supports a "Base URI Embedded in Content" in the form of base elements. A base element applies to all subsequent elements in a document until the next base element is encountered. The URI reference in a base element itself is resolved relative to the base URI of next lower precedence (which is usually the "URI used to retrieve the entity" but can be the "Base URI of the encapsulating entity").

5.2. Transform References

The following pseudocode describes an algorithm for transforming a URI reference R into its target URI T using the base URI B, the href type H, and the link or form relation type S. The URI reference and base URI are assumed to be pre-parsed into a sequence of Href.* options; the result is returned as a sequence of Href.* options as well.

```
if (R starts with Href.Scheme) then
   T = R
elif (R starts with Href.Host.*) then
   T = [ (k, v) | (k, v) <- B, k == Href.Scheme ] ++
       [ (k, v) | (k, v) <- R, k >  Href.Scheme ]
elif (R starts with Href.Port) then
   T = [ (k, v) | (k, v) <- B, k <  Href.Port ] ++
       [ (k, v) | (k, v) <- R, k >= Href.Port ]
elif (H is append-relation) then
   T = [ (k, v) | (k, v) <- B, k <= Href.Path ] ++
       [ (Href.Path, (hex S)) ]
elif (H is append-path) then
   T = [ (k, v) | (k, v) <- B, k <= Href.Path ] ++
       [ (k, v) | (k, v) <- R, k >= Href.Path ]
elif (H is relative-path) then
   T = [ (k, v) | (k, v) <- B, k <  Href.Path ] ++
 (init [ (k, v) | (k, v) <- B, k == Href.Path ]) ++
       [ (k, v) | (k, v) <- R, k >= Href.Path ]
elif (H is absolute-path) then
   T = [ (k, v) | (k, v) <- B, k <  Href.Path ] ++
       [ (k, v) | (k, v) <- R, k >= Href.Path ]
endif
```

The "init" function returns all the elements of the input list except the last one. For example, (init [1, 2, 3]) returns [1, 2] and (init []) returns [].

The "hex" function returns a hexadecimal representation of the input number. For example, (hex -421) returns "-1A5" and (hex 0) returns "0".

5.3. Remove Dot Segments

After transforming a the URI reference into its target URI, the special path segments "." and ".." need to be removed. Although there are many ways to accomplish this removal process, we describe a simple method using two string buffers.

1. The input buffer is initialized with the sequence of path segments and the output buffer is initialized to the empty sequence.

2. While the input buffer is not empty, loop as follows:

   * If the input buffer begins with a "." path segment, then remove this segment from the input buffer; otherwise,

   * if the input buffer begins with a ".." path segment, then remove this segment from the input buffer and and remove the last segment (if any) from the output buffer; otherwise,

   * move the first path segment in the input buffer to the end of the output buffer.

3. Finally, the sequence of path segments in the target URI is replaced by the sequence of path segments in the output buffer.

6. Profiles

Profiles provide an easy way to extend CoRAL with application-specific link relation types and form relation types.

Application-specific link relation types and form relation types are encoded in CoRAL as negative numbers. The meaning of these numbers is defined by the profile, which maps them to extension link and form relation types. (An extension relation type is a URI [RFC3986] that uniquely identifies the relation type; see Section 4.2 of [RFC5988].) For example, a CoRAL profile for online shops could define the following set of application-specific link relation types:

ID	Link Relation Type
-80	http://example.com/rels/order
-81	http://example.com/rels/basket
-82	http://example.com/rels/customer

Table 3: Example profile for online shops

A good source for existing extension link relation types are RDF predicates [W3C.REC-rdf11-concepts-20140225]. An RDF statement says that some relationship, indicated by a predicate, holds between two resources. Link relation types and RDF predicates can therefore be used interchangeably in many cases. For example, a CoRAL profile could define the following mapping from link relation type numbers to the FOAF vocabulary [FOAF]:

ID	RDF Predicate
-100	http://xmlns.com/foaf/0.1/name
-101	http://xmlns.com/foaf/0.1/age
-102	http://xmlns.com/foaf/0.1/homepage

Table 4: Example profile for the FOAF vocabulary

CoRAL profiles are identified by a URI [RFC3986] and are indicated by the 'profile' parameter of the 'application/coral' media type, following the recommendations of [RFC6906], Section 3.1. For example, the FOAF profile above could use the media type 'application/coral; profile="http://xmlns.com/foaf/0.1/"'.

The profile URI serves only as an identifier, similar to an XML namespace URI [W3C.REC-xml-names-20091208]. That is, the profile URI does not necessarily have to identify a dereferencable resource (or even use a dereferencable URI scheme). It is possible, though, to use a dereferencable URI and serve a representation that provides information about the profile 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 use only at most one profile at the same time. The 'profile' parameter of the 'application/coral' media type contains a single profile URI, not a whitespace-separated list of profile URIs as recommended by [RFC6906].

Profile URIs do not have to be registered. The use of DNS names in them allows decentralized creation of new profile identifiers without the risk of collisions. However, in CoAP the media type of a representation is indicated by a small numeric identifier, called the content format, not by a string of characters. A content format has no internal structure and indicates a media type including specific values for its parameters. For example, content format 0 indicates the media type 'text/plain; charset=utf-8'. A media type 'text/plain; charset=iso-8859-1' would require a separate content format. Therefore, for use with CoAP, each CoRAL profiles needs to register a new content format in the "CoAP Content-Formats" registry, specifying the 'application/coral' media type and the profile URI as the value of the 'profile' parameter.

7. Security Considerations

TODO.

8. IANA Considerations

8.1. Media Type

This document registers the media type 'application/coral' in the Media Types Registry.

Type name:
application

Subtype name:
coral

Required parameters:
N/A

Optional parameters:

profile
See Section 6 of [RFCXXXX].

Encoding considerations:
CoRAL is a binary encoding.

Security considerations:
See Section 7 of [RFCXXXX].

Interoperability considerations:
There are no known interoperability issues.

Published specification:
[RFCXXXX]

Applications that use this media type:
Hypermedia-driven Web applications that run on constrained nodes and networks.

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 "Author's Address" section of [RFCXXXX].

Intended usage:
COMMON

Restrictions on usage:
N/A

Author:
See "Author's Address" section of [RFCXXXX].

Change controller:
IESG

8.2. Content Format

This document registers a content format for the 'application/coral' media type in the CoAP Content-Formats Registry.

Media type:
application/coral

Content coding:
identity

ID:
TBD

Reference:
[RFCXXXX]

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6906] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <http://www.rfc-editor.org/info/rfc6906>.

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

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

9.2. Informative References

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

[I-D.greevenbosch-appsawg-cbor-cddl] Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-09 (work in progress), September 2016.

[I-D.hartke-core-apps]
Hartke, K., "CoRE Application Descriptions", draft-hartke-core-apps-07 (work in progress), February 2017.

[I-D.hartke-core-lighting]
Hartke, K., "CoRE Lighting", draft-hartke-core-lighting-00 (work in progress), September 2015.

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., Koster, M., and C. Groves, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-08 (work in progress), February 2017.

[I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-06 (work in progress), July 2016.

[I-D.kelly-json-hal]
Kelly, M., "JSON Hypertext Application Language", draft-kelly-json-hal-08 (work in progress), May 2016.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://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, <http://www.rfc-editor.org/info/rfc7228>.

[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>.

[W3C.REC-xml-names-20091208]
Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

Appendix A. Examples

A.1. Overview

A.1.1. Links

At its core, CoRAL is just yet another serialization format for Web links. For example, the following Web link (in RFC 5988 syntax):

```
<coap://example.com/info/tos>;rel=terms-of-service;type=text/plain
```

can be serialized in CoRAL as the following CBOR data item (in CBOR extended diagnostic format; text enclosed in forward slashes are comments):

```
[ [ 5,                    / fat-link                          /
    1,                    / absolute-path                     /
    [  1, 70,             /  relation       = terms-of-service  /
       4, 0,              /  format         = text\plain        /
       5, "coap",         /  href.scheme    = "coap"            /
       6, "example.com",  /  href.host.name = "example.com"     /
      10, "info",         /  href.path      = "info"            /
      10, "tos" ] ] ]     /  href.path      = "tos"             /
```

Multiple links are serialized as items of the top-level array:

```
[ [ 5,                    / fat-link                          /
    3,                    / relative-path                     /
    [  1, 26,             /  relation  = first                 /
       4, 0,              /  format    = text\plain           /
      10, "page1" ] ],    /  href.path = "page1"              /
  [ 5,                    / fat-link                          /
    3,                    / relative-path                     /
    [  1, 55,             /  relation  = previous              /
       4, 0,              /  format    = text\plain           /
      10, "page6" ] ],    /  href.path = "page6"              /
  [ 5,                    / fat-link                          /
    3,                    / relative-path                     /
    [  1, 41,             /  relation  = next                  /
       4, 0,              /  format    = text\plain           /
      10, "page8" ] ],    /  href.path = "page8"              /
  [ 5,                    / fat-link                          /
    3,                    / relative-path                     /
    [  1, 34,             /  relation  = last                  /
       4, 0,              /  format    = text\plain           /
      10, "page42" ] ] ]  /  href.path = "page42"             /
```

The Format Option, when present, is a hint indicating what the CoAP content format of the result of dereferencing the link should be. If more than one format is available, the Format Option can be repeated:

```
[ [ 5,                    / fat-link                          /
    3,                    / relative-path                     /
    [  1, 33,             /  relation  = item                  /
       4, 47,             /  format    = application\exi      /
       4, 50,             /  format    = application\json     /
       4, 60,             /  format    = application\cbor     /
      10, "item1" ] ] ]   /  href.path = "item1"              /
```

A.1.2. Embedding

If a representation links to many resources, it may be inefficient to retrieve a representation of each link target individually. For this reason, CoRAL supports the embedding of a representation of the link target in the link itself:

```
    [ [ 5,                    / fat-link                          /
        3,                    / relative-path                     /
        [  1, 33,             /  relation  = item                  /
           4, 50,             /  format    = application\json      /
          10, "item1" ],      /  href.path = "item1"               /
        h'7b20227461736b223a20
          2252657475726e207468
          6520626f6f6b7320746f
          20746865206c69627261
          7279222c202261737369
          676e6565223a2022416c
          69636522207d' ] ]
```

where the byte string in this example encodes the following JSON object:

```
{
  "task":     "Return the books to the library",
  "assignee": "Alice"
}
```

By embedding representations, it is possible to use CoRAL as a (very basic) substitute for RDF [W3C.REC-rdf11-concepts-20140225]. For example, the RDF graph (in Turtle [W3C.REC-turtle-20140225] syntax):

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<> foaf:name     "John Doe" ;
   foaf:age      32 ;
   foaf:homepage <coap://www.doe.example/> .
```

can be serialized in CoRAL as follows:

```
[ [ 2,                    / tiny-literal                       /
    2,                    / append-path                        /
    -100,                 /  relation       = name             /
    0,                    /  format         = text\plain       /
    h'4a6f686e20446f65' ], / "John Doe"                        /
  [ 2,                    / tiny-literal                       /
    2,                    / append-path                        /
    -101,                 /  relation       = age              /
    7,                    /  format         = uint8            /
    h'20' ],              / 32                                 /
  [ 5,                    / fat-link                           /
    1,                    / absolute-path                      /
    [ 1, -102,            /  relation       = homepage         /
      5, "coap",          /  href.scheme    = "coap"           /
      6, "www.doe.example" /  href.host.name = "www.doe.example" /
] ] ]
```

A.1.3. Forms

In addition to Web links, CoRAL supports forms. An agent can use a form to perform an operation on the form context, such as updating a resource or creating a new item in a collection.

Similar to link relation types, the semantics of a form are indicated by the form relation type. The Href.* Options encode the URI of the form target to which the agent should submit the form. A form additionally encodes the submission method (POST, PUT, PATCH, DELETE) and the description of a representation that the service expects as part of form submission:

```
[ [ 7,                    / fat-form                           /
    3,                    / relative-path                      /
    [  1, 1,              /  relation  = create-item            /
       2, 2,              /  method    = POST                  /
       3, 60,             /  accept    = application\cbor      /
      10, "items" ] ] ]   /  href.path = "items"               /
```

The Accept Option specifies the content format of the expected representation. A content format can use the body of a form to describe the expected representation in more detail, for example, by specifying a set of form fields that the agent needs to fill out:

```
[ [ 7,                    / fat-form                          /
    3,                    / relative-path                     /
    [  1, 1,              /  relation  = create-item           /
       2, 2,              /  method    = POST                 /
       3, 65535,          /  accept    = example\form         /
      10, "items" ],      /  href.path = "items"              /
    h'6e616d652c206167652c / "name, age, homepage"            /
      20686f6d6570616765' ] ]
```

A.1.4. Editing

The target resource of a link may be editable. In this case, the representation of such a resource typically contains one or more forms that allow an agent to edit the resource. However, it may be inefficient to include these forms every time a representation of the link target is retrieved and more efficient to include them in representations that link to that resource. CoRAL supports this with two options.

Setting the Updatable Option in a link to true defines a form that can be used to update the target resource. The context and target of that form are both the target of the link, the submission method is PUT and the content format of the submitted representation must be one of the formats indicated by the Format Option in the link. For example, given the following CoRAL representation, an agent can change the recipient by making a PUT request to <./to> with the new value in 'text/plain' format:

```
  [ [ 5,                    / fat-link                          /
      3,                    / relative-path                     /
      [  1, -120,           /  relation  = sender               /
         4, 0,              /  format    = text\plain          /
        10, "from",         /  href.path = "from"              /
        14, true ],         /  updatable = true                /
      h'4a756c696574' ],    / "Juliet"                          /
    [ 5,                    / fat-link                          /
      3,                    / relative-path                     /
      [  1, -121,           /  relation  = recipient            /
         4, 0,              /  format    = text\plain          /
        10, "to",           /  href.path = "to"                /
        14, true ],         /  updatable = true                /
      h'526f6d656f' ],      / "Romeo"                           /
    [ 5,                    / fat-link                          /
      3,                    / relative-path                     /
      [  1, -122,           /  relation  = message              /
         4, 0,              /  format    = text\plain          /
        10, "message",      /  href.path = "message"           /
        14, true ],         /  updatable = true                /
      h'4172742074686f75206e / "Art thou not Romeo,            /
        6f7420526f6d656f2c20 /  and a Montague?"                /
        616e642061204d6f6e74
        616775653f' ] ]
```

Setting the Deletable Flag in a link to true likewise defines a form that can be used to delete the target resource.

A.2. CoRE Lighting

CoRE Lighting [I-D.hartke-core-lighting] defines a benchmark scenario for the exploration of hypermedia-oriented design in constrained, RESTful environments. The bulletin board example in Section 5.2.1 of [I-D.hartke-core-lighting] can be encoded in CoRAL as follows:

```
  [[7, 3, [1, 1, 3, 65200, 10, "bulletins"]],
   [5, 1, [1, 33, 4, 65200, 6, "light-bulb.example"], <<1>>],
   [5, 1, [1, 33, 4, 65200, 6, "remote-control.example"], <<2>>]]
```

where <<1>> is a byte string that encodes the following CoRAL structure:

```
  [[5, 3, [1, -100, 4, 65202, 10, "config"]],
   [2, 3, -101, 0, "Light 2"],
   [2, 3, -102, 0, "Illuminates the couch."],
   [2, 3, -103, 0, "Living Room"]]
```

and <<2>> is a byte string that encodes the following CoRAL structure:

```
  [[5, 3, [1, 1, 4, 65203, 10, "state"]],
   [2, 3, -101, 0, "LRC 1"],
   [2, 3, -102, 0, "Controls Light 2."],
   [2, 3, -103, 0, "Living Room"]]
```

Table 5 shows a comparison of sizes of the example encoded in CoRAL and JSON.

Format	Size
JSON	515 bytes
CoRAL	245 bytes

Table 5: Size Comparison

A.3. CoRE Link Format

The example in this section is based on an example on page 14 of [RFC6690]:

```
   </sensors>;ct=40;title="Sensor Index",
   </sensors/temp>;rt="temperature-c";if="sensor",
   </sensors/light>;rt="light-lux";if="sensor",
   <http://www.example.com/sensors/t123>;anchor="/sensors/temp"
   ;rel="describedby",
   </t>;anchor="/sensors/temp";rel="alternate"
```

The example can be encoded in CoRAL as follows:

```
  [[4, 1, [10, "sensors"]],
   [5, 2, [4, 40, 13, "Sensor Index"]],
   [5, 2, [10, "temp", 16, "temperature-c", 17, "sensor"]],
   [5, 2, [10, "light", 16, "light-lux", 17, "sensor"]],
   [5, 1, [1, 18, 5, "http", 6, "www.example.com",
           10, "sensors", 10, "t123", 18, "/sensors/temp"]],
   [5, 1, [1, 2, 10, "t", 18, "/sensors/temp"]]]
```

Table 6 shows a comparison of sizes of the example encoded in CoRAL and a number of Link Format variants [I-D.ietf-core-links-json].

Format	Size
Link Format	251 bytes
Link Format (JSON)	320 bytes
Link Format (CBOR)	203 bytes
CoRAL	181 bytes

Table 6: Size Comparison

A.4. CoRE Interfaces

The example in this section is based on an example in figure 1 of [I-D.ietf-core-interfaces]:

```
</s/>;rt="simple.sen";if="core.b",
</s/lt>;rt="simple.sen.lt";if="core.s",
</s/tmp>;rt="simple.sen.tmp";if="core.s";obs,
</s/hum>;rt="simple.sen.hum";if="core.s",
</a/>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d/>;rt="simple.dev";if="core.ll",
</l/>;if="core.lb"
```

The example can be encoded in CoRAL as follows:

```
[[5, 1, [10, "d", 10, "", 16, "simple.dev", 17, "core.ll"]],
 [5, 1, [10, "l", 10, "", 17, "core.lb"]],
 [4, 1, [17, "core.b"]],
 [5, 2, [10, "s", 10, "", 16, "simple.sen"]],
 [5, 2, [10, "a", 10, "", 16, "simple.act"]],
 [4, 1, [10, "s", 17, "core.s"]],
 [5, 2, [10, "lt", 16, "simple.sen.lt"]],
 [5, 2, [10, "tmp", 16, "simple.sen.tmp", 19, true]],
 [5, 2, [10, "hum", 16, "simple.sen.hum"]],
 [4, 1, [10, "a", 16, "simple.act.led", 17, "core.a"]],
 [5, 2, [10, "1", 10, "led"]],
 [5, 2, [10, "2", 10, "led"]]]
```

Table 7 shows a comparison of sizes of the example encoded in CoRAL and a number of Link Format variants.

Format	Size
Link Format	332 bytes
Link Format (JSON)	456 bytes
Link Format (CBOR)	264 bytes
CoRAL	248 bytes

Table 7: Size Comparison

Acknowledgements

This specification is heavily inspired by the JSON Hypertext Application Language (HAL) [I-D.kelly-json-hal]; the author of and contributors to that specification are acknowledged for their great work.

Yassin Nasir Hassan suggested placing the hypermedia controls for modifying a link target in the link context rather than in the representation of the link target.

Thanks to Carsten Bormann, Jaime Jimenez and Matthias Kovatsch for helpful comments and discussions that have shaped the document.

Author's Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org