The Constrained RESTful Application Language (CoRAL)
draft-hartke-t2trg-coral-01

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 April 20, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
Table of Contents

1. Introduction ........................................... 2
   1.1. Terminology ..................................... 3
2. Model ................................................... 3
3. Format .................................................. 5
4. Options ................................................. 6
   4.1. Accept ........................................... 8
   4.2. Deletable .......................................... 8
   4.3. Format ............................................. 8
   4.4. Href.* ............................................ 8
   4.5. Method ............................................. 10
   4.6. Relation ........................................... 10
   4.7. Title ............................................... 10
   4.8. Updatable ......................................... 10
5. Reference Resolution .................................... 10
   5.1. Establish a Base URI ............................. 10
   5.2. Transform References ............................ 11
   5.3. Remove Dot Segments .............................. 12
6. Security Considerations ................................ 12
7. IANA Considerations ................................... 12
   7.1. CoRAL Option Number Registry .................. 12
   7.2. Media Type ........................................ 14
   7.3. Structured Syntax Suffix ....................... 15
   7.4. CoAP Content-Format ............................. 16
8. References ............................................... 16
   8.1. Normative References ............................ 16
   8.2. Informative References .......................... 17
Appendix A. Examples ....................................... 18
   A.1. Overview ......................................... 18
   A.2. CoRE Lighting ..................................... 23
   A.3. CoRE Link Format .................................. 24
   A.4. CoRE Interfaces ................................... 25
Acknowledgements .......................................... 26
Author’s Address .......................................... 26

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 the 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 that 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 two. 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 in CoRAL again 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, in the form of 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. 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 can decide again what to do next.

An agent can furthermore navigate a browsing context by traversing the browsing context’s session history. The session history consists...
of a flat list of session history entries. Each session history entry consists of a URI and may have other information associated with it. Session history entries are added to the session history as the agent navigates from resource to resource. An agent can traverse the session history to any entry 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" or a media type derived from CoRAL with the structured syntax suffix "+coral". CoRAL is typically embedded in CBOR-based media types, but other media types can embed CoRAL as well. The CoRAL format is in all cases the same.

The top-level structure of CoRAL is a CoRAL document. A CoRAL document consists of a sequence of links, forms, literals and bases, which are collectively called elements. 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. 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 (non-negative) numeric identifiers instead of the textual names. The same optimization is applied to form relation types, CoAP content formats and CoAP request methods.

Applications can use negative numbers to indicate application-specific link relation types, form relation types and content formats, which do not need to be IANA-registered. The mappings from numbers to textual names need to be provided by the respective media type definition (i.e., a media type with the "+coral" suffix or a media type embedding CoRAL; the "application/coral" media type itself does not define any application-specific values).
CoRAL defines a number of options that can be included in link, form, literal and base elements. Options are used to encode the relation type, the target resource URI and target attributes. They are serialized as a sequence of unwrapped pairs where each pair consists of a CoRAL option number and an option value. The pairs MUST be sorted such that the option numbers are in ascending order.

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

\[
\begin{align*}
document & = [*element] \\
element & = tiny-link / tiny-literal / tiny-form \\
& / fat-link / fat-literal / fat-form \\
& / base \\
tiny-link & = [1, href-type, relation] \\
tiny-literal & = [2, href-type, relation, format, body] \\
tiny-form & = [3, href-type, relation, accept] \\
base & = [4, href-type, options] \\
fat-link & = [5, href-type, options, ?body] \\
fat-literal & = [6, href-type, options, body] \\
fat-form & = [7, href-type, options, ?body] \\
href-type & = &{(append-relation: 0, \\
& absolute-path: 1, \\
& append-path: 2, \\
& relative-path: 3)} \\
relation & = int \\
format & = int \\
accept & = int \\
options & = [*{option-number, option-value}] \\
option-number & = uint \\
option-value & = uint / int / text / bytes \\
body & = bytes \\
\end{align*}
\]

The tiny formats expand as follows:

\[
\begin{align*}
[1, H, R] & \rightarrow [5, H, [1, R]] \\
\end{align*}
\]

4. Options

Table 1 summarizes the CoRAL options defined in this document.
<table>
<thead>
<tr>
<th>No.</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>Relation</td>
<td>int</td>
<td>(none)</td>
<td>(none)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Method</td>
<td>uint</td>
<td></td>
<td>2 (POST)</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>Accept</td>
<td>int</td>
<td>(none)</td>
<td>(none)</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>Format</td>
<td>int</td>
<td>(none)</td>
<td>(none)</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Href.Scheme</td>
<td>text</td>
<td>1-255</td>
<td>(none)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Href.Host.Name</td>
<td>text</td>
<td>1-255</td>
<td>(none)</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Href.Host.IPv4</td>
<td>bytes</td>
<td>4</td>
<td>(none)</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Href.Host.IPv6</td>
<td>bytes</td>
<td>16</td>
<td>(none)</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>Href.Port</td>
<td>uint</td>
<td></td>
<td>(see below)</td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td>Href.Path</td>
<td>text</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>11</td>
<td>x</td>
<td>Href.Query</td>
<td>text</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>Href.Fragment</td>
<td>text</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>Title</td>
<td>text</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>Updatable</td>
<td>bool</td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>Deletable</td>
<td>bool</td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>

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 individual options are explained in the following subsections.
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 either one of the content format IDs registered in the CoAP Content-Formats registry (>= 0) or one of the application-specific content format IDs defined by the media type (< 0).

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 context and target of that form are the link target, 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 can set the Accept Option in its CoAP request 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.

The option value of a Format Option is either one of the content format IDs registered in the CoAP Content-Formats registry (>= 0) or one of the application-specific content format IDs defined by the media type (< 0).

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,
o the Href.Host.Name Option specifies the host as a registered name [RFC3986],

o the Href.Host.IPv4 Option specifies the host as a 32-bit IPv4 address,

o the Href.Host.IPv6 Option specifies the host as a 128-bit IPv6 address,

o the Href.Port Option specifies the port number,

o each Href.Path Option specifies one segment of the path,

o each Href.Query Option specifies one argument of the query, and

o 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.

<table>
<thead>
<tr>
<th></th>
<th>absolute-path</th>
<th>relative-path</th>
<th>append-path</th>
<th>append-relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Href.Scheme</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Href.Host.Name</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Href.Host.IPv4</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Href.Host.IPv6</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Href.Port</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Href.Path</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Href.Query</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Href.Fragment</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

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.

The option value of a Relation Option is either one of the relation type IDs registered in the Link Relation Types registry (>= 0) or one of the application-specific relation type IDs defined by the media type (< 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 context and target of that form are the link target, 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.

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

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

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

   TODO.

7. IANA Considerations

7.1. CoRAL Option Number Registry

   This document establishes the CoRAL Option Number registry for the option numbers used in CoRAL. The registry is located within the CoRE Parameters registry.

7.1.1. Registering New Option Numbers

   Option numbers are registered on the advice of a Designated Expert (appointed by the IESG or their delegate), with a Specification Required (using terminology from [RFC5226]).

   Registration requests consist of the completed registration template below, typically published in an RFC. However, to allow for the allocation of values prior to publication, the Designated Expert may
approve registration once they are satisfied that a specification will be published.

The registration template is:

- Option Number:
- Option Name:
- Reference:

### 7.1.2. Initial Registry Contents

The CoRAL Option Number registry’s initial contents are:

- Option Number: 1
  - Option Name: Relation
  - Reference: [RFCXXXX]

- Option Number: 2
  - Option Name: Method
  - Reference: [RFCXXXX]

- Option Number: 3
  - Option Name: Accept
  - Reference: [RFCXXXX]

- Option Number: 4
  - Option Name: Format
  - Reference: [RFCXXXX]

- Option Number: 5
  - Option Name: Href.Scheme
  - Reference: [RFCXXXX]

- Option Number: 6
  - Option Name: Href.Host.Name
  - Reference: [RFCXXXX]

- Option Number: 7
  - Option Name: Href.Host.IPv4
  - Reference: [RFCXXXX]

- Option Number: 8
  - Option Name: Href.Host.IPv6
  - Reference: [RFCXXXX]

- Option Number: 9
Option Name: Href.Port  
Reference: [RFCXXXX]

Option Number: 10  
Option Name: Href.Path  
Reference: [RFCXXXX]

Option Number: 11  
Option Name: Href.Query  
Reference: [RFCXXXX]

Option Number: 12  
Option Name: Href.Fragment  
Reference: [RFCXXXX]

Option Number: 13  
Option Name: Title  
Reference: [RFCXXXX]

Option Number: 14  
Option Name: Updatable  
Reference: [RFCXXXX]

Option Number: 15  
Option Name: Deletable  
Reference: [RFCXXXX]

7.2. 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:  
N/A

Encoding considerations:  
CoRAL is a binary encoding.

Security considerations:
See Section 6 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 in 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

7.3. Structured Syntax Suffix

This document registers the suffix "+coral" in the "Structured Syntax Suffix" registry.

Name:
Constrained RESTful Application Language (CoRAL)
Internet-Draft                    CoRAL                     October 2016

+suffix:
  +coral

References:
  [RFCXXXX]

Encoding considerations:
  CoRAL is a binary format.

Interoperability considerations:
  There are no known interoperability issues.

Fragment identifier considerations:
  The syntax and semantics of fragment identifiers specified for
  +coral are as specified for "application/coral". (At publication
  of this document, there is no fragment identification syntax
  defined for "application/coral".)

Security considerations:
  See Section 6 of [RFCXXXX].

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

Author/Change controller:
  IESG

7.4.  CoAP Content-Format

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

 o Media Type: application/coral
   Encoding: -
   ID: 70
   Reference: [RFCXXXX]

8.  References

8.1.  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,
8.2. Informative References


Appendix A. Examples

A.1. Overview

A.1.1. Links

At its core, CoRAL is 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; forward slashes indicate comments):

```cbor
[[ 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:

```cbor
[[ 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:

```cbor
[[ 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
  6520626f6b7320746f
  207468652074686520626f6f6b7320746f
  20746865206c69627279222c2022617373696565223a2022416c69636522207d' 
]
```

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:
A.1.3. Namespaces

The link relation type in a serialized link may be from the "global" or the "local" namespace. The global namespace is indicated by an unsigned number and is made up of the link relation types registered with IANA. The local namespace is indicated by a negative number and is defined by the media type of the CoRAL representation.

By default, CoRAL representations have the "application/coral" media type where the local namespace is empty. However, it is possible to create new media types based on CoRAL and to register these with the "+coral" suffix. In this case, the media type specification can fill the local namespace with application-specific link relation types.

For example, a media type "application/example.shop+coral" could define the following set of local link relation types:

```
+-----+----------------------------------+
| ID  | Meaning                          |
+-----+----------------------------------+
| -80 | http://example.com/rels/order    |
| -81 | http://example.com/rels/basket   |
| -82 | http://example.com/rels/customer |
+-----+----------------------------------+
```

Similarly, a media type "application/example.foaf+coral" could define the following mapping from link relation type IDs to the FOAF RDF model [FOAF]:

```
+-----+----------------------------------+
| ID  | Meaning                          |
+-----+----------------------------------+
| -80 | http://example.com/rels/order    |
| -81 | http://example.com/rels/basket   |
| -82 | http://example.com/rels/customer |
| 5   | "coap",                          |
| 6   | "www.doe.example"                |
+-----+----------------------------------+
```

### A.1.4.  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" / 20686f6d5706167652c ] ]
```

### A.1.5.  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 the 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,          /
  6f74205266d656f2c20 /  and a Montague?"                   /
  616642061204d6f6e74 61675653f' ] ]
```

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:
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 3 shows a comparison of sizes of the example encoded in CoRAL and JSON.

```
+--------+-----------+
<table>
<thead>
<tr>
<th>Format</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td>515 bytes</td>
</tr>
<tr>
<td>CoRAL</td>
<td>245 bytes</td>
</tr>
</tbody>
</table>
+--------+-----------+
```

Table 3: Size Comparison

A.3. CoRE Link Format

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

```html
</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:
Table 4 shows a comparison of sizes of the example encoded in CoRAL and a number of Link Format variants.

<table>
<thead>
<tr>
<th>Format</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link Format</td>
<td>251 bytes</td>
</tr>
<tr>
<td>Link Format (JSON)</td>
<td>320 bytes</td>
</tr>
<tr>
<td>Link Format (CBOR)</td>
<td>203 bytes</td>
</tr>
<tr>
<td>CoRAL</td>
<td>181 bytes</td>
</tr>
</tbody>
</table>

Table 4: Size Comparison

A.4. CoRE Interfaces

The example in this section is based on an example on page 10 of [I-D.ietf-core-interfaces]:

```html
<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:
Table 5 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 5: 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.

Yassine 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 Bornann, Jaime Jimenez and Matthias Kovatsch for helpful comments and discussions that have shaped the document.
Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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