

Thing-to-Thing Research Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

K. Hartke
Ericsson
October 22, 2018

CoRE Applications
draft-hartke-core-apps-08

Abstract

The application programmable interfaces of RESTful, hypermedia-driven Web applications consist of a number of reusable components such as Internet media types and link relation types. This document proposes "CoRE Applications", a convention for application designers to build the interfaces of their applications in a structured way, so that implementers can easily build interoperable clients and servers, and other designers can reuse the components in their own applications.

Note to Readers

This Internet-Draft should be discussed on the Thing-to-Thing Research Group (T2TRG) mailing list <t2trg@irtf.org> <<https://www.irtf.org/mailman/listinfo/t2trg>>.

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 April 25, 2019.

Copyright Notice

Copyright (c) 2018 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 (<https://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
2. CoRE Applications	3
2.1. Communication Protocols	4
2.1.1. URI Schemes	4
2.2. Representation Formats	5
2.2.1. Internet Media Types	5
2.3. Links	7
2.3.1. Link Relation Types	8
2.3.2. Template Variable Names	8
2.4. Forms	8
2.4.1. Form Relation Types	9
2.4.2. Form Field Names	9
2.5. Well-Known Locations	10
3. CoRE Application Descriptions	10
3.1. Template	11
4. URI Design Considerations	12
5. Security Considerations	14
6. IANA Considerations	14
7. References	14
7.1. Normative References	14
7.2. Informative References	15
Acknowledgements	16
Author's Address	17

1. Introduction

Representational State Transfer (REST) [16] is an architectural style for distributed hypermedia systems. Over the years, REST has gained popularity not only as an approach for large-scale information dissemination, but also as the basic principle for designing and building Internet-based applications in general.

In the coming years, the size and scope of the Internet is expected to increase greatly as physical-world objects become smart enough to communicate over the Internet -- a phenomenon known as the Internet of Things (IoT). As things learn to speak the languages of the net,

the idea of applying REST principles to the design of IoT application architectures suggests itself. To this end, the Constrained Application Protocol (CoAP) [23] was created, an application-layer protocol that enables RESTful applications in constrained-node networks [10], giving rise to a new setting for Internet-based applications: the Constrained RESTful Environment (CoRE).

To realize the full benefits and advantages of the REST architectural style, a set of constraints needs to be maintained when designing applications and their application programming interfaces (APIs). One of the fundamental principles of REST is that "REST APIs must be hypertext-driven" [17]. However, this principle is often ignored by application designers. Instead, APIs are specified out-of-band in terms of fixed URI patterns (e.g., in the API documentation or in a machine-readable format that facilitates code generation). Although this approach may appear easy for clients to use, the fixed resource names and data formats lead to a tight coupling between client and server implementations and make the system less flexible [5]. Violations of REST design principles like this result in APIs that may not be as scalable, extensible, and interoperable as promised by REST.

REST is intended for network-based applications that are long-lived and span multiple organizations [17]. Principled REST APIs require some design effort, since application designers do not only have to take current requirements into consideration, but also have to anticipate changes that may be required in the future -- years or even decades after the application has been deployed for the first time. The reward is long-term stability and evolvability, both of which are very desirable features in the Internet of Things.

To aid application designers in the design process, this document proposes "CoRE Applications", a convention for building the APIs of RESTful, hypermedia-driven Web applications. The goal is to help application designers avoid common mistakes by focusing almost all of the descriptive effort on defining the Internet media type(s) that are used for representing resources and driving application state.

A template for a "CoRE Application Description" provides a consistent format for the description of APIs so that implementers can easily build interoperable clients and servers, and other application designers can reuse the components in their own applications.

2. CoRE Applications

A CoRE Application API is a named set of reusable components. It describes a contract between a server hosting an instance of the

described application and clients that wish to interface with that instance.

The API is generally comprised of:

- o communication protocols, identified by URI schemes,
- o representation formats, identified by Internet media types,
- o link relation types,
- o form relation types,
- o template variables in templated links,
- o form field names in forms, and
- o well-known locations.

Together, these components provide the specific, in-band instructions to a client for interfacing with a given application.

2.1. Communication Protocols

The foundation of a hypermedia-driven REST API are the communication protocol(s) spoken between a client and a server. Although HTTP/1.1 [14] is by far the most common communication protocol for REST APIs, a REST API should typically not be dependent on any specific communication protocol.

2.1.1. URI Schemes

The usage of a particular protocol by a client is guided by URI schemes [7]. URI schemes specify the syntax and semantics of URI references [1] that the server includes in hypermedia controls such as links and forms.

A URI scheme refers to a family of protocols, typically distinguished by a version number. For example, the "http" URI scheme refers to the two members of the HTTP family of protocols: HTTP/1.1 [14] and HTTP/2 [8] (as well as some predecessors). The specific HTTP version used is negotiated between a client and a server in-band using the version indicator in the HTTP request-line or the TLS Application-Layer Protocol Negotiation (ALPN) extension [18].

IANA maintains a list of registered URI schemes at <http://www.iana.org/assignments/uri-schemes>.

2.2. Representation Formats

In RESTful applications, clients and servers exchange representations that capture the current or intended state of a resource and that are labeled with a media type. A representation is a sequence of bytes whose structure and semantics are specified by a representation format: a set of rules for encoding information.

Representation formats should generally allow clients with different goals, so they can do different things with the same data. The specification of a representation format "describes a problem space, not a prescribed relationship between client and server. Client and server must share an understanding of the representations they're passing back and forth, but they don't need to have the same idea of what the problem is that needs to be solved." [21]

Representation formats and their specifications frequently evolve over time. It is part of the responsibility of the designer of a new version to insure both forward and backward compatibility: new representations should work reasonably (with some fallback) with old processors and old representations should work reasonably with new processors [20].

Representation formats enable hypermedia-driven applications when they support the expression of hypermedia controls such as links (Section 2.3) and forms (Section 2.4).

2.2.1. Internet Media Types

One of the most important aspect of hypermedia-driven communications is the concept of Internet media types [2]. Media types are used to label representations so that it is known how the representation should be interpreted and how it is encoded. The centerpiece of a CoRE Application Description should be one or more media types.

Note: The terms media type and representation format are often used interchangeably. In this document, the term "media type" refers specifically to a string of characters such as "application/xml" that is used to label representations; the term "representation format" refers to the definition of the syntax and semantics of representations, such as XML 1.0 [12] or XML 1.1 [13].

A media type identifies a versioned series of representation formats (Section 2.2): a media type does not identify a particular version of a representation format; rather, the media type identifies the family, and includes provisions for version indicator(s) embedded in the representations themselves to determine more precisely the nature

of how the data is to be interpreted [20]. A new media type is only needed to designate a completely incompatible format [20].

Media types consist of a top-level type and a subtype, structured into trees [2]. Optionally, media types can have parameters. For example, the media type "text/plain; charset=utf-8" is a subtype for plain text under the "text" top-level type in the standards tree and has a parameter "charset" with the value "utf-8".

Media types can be further refined by

- o structured type name suffixes (e.g., "+xml" appended to the base subtype name; see Section 4.2.8 of RFC 6838 [2]),
- o a "profile" parameter (see Section 3.1 of RFC 6906 [24]),
- o subtype information embedded in the representations themselves (e.g., "xmlns" declarations in XML documents [11]),

or a similar annotation. An annotation directly in the media type is generally preferable, since subtype information embedded in representations can typically not be negotiated during content negotiation (e.g., using the CoAP Accept option).

In CoAP, media types are paired with a content coding [15] to indicate the "content format" [23] of a representation. Each content format is assigned a numeric identifier that can be used instead of the (more verbose) textual name of the media type in representation formats with size constraints. The flat number space loses the structural information that the textual names have, however.

The media type of a representation must be determined from in-band information (e.g., from the CoAP Content-Format option). Clients must not assume a structure from the application context or other out-of-band information.

IANA maintains a list of registered Internet media types at <http://www.iana.org/assignments/media-types>.

IANA maintains a list of registered structured suffixes at <http://www.iana.org/assignments/media-type-structured-suffix>.

IANA maintains a list of registered CoAP content formats at <http://www.iana.org/assignments/core-parameters>.

2.3. Links

As defined in RFC 8288 [6], a link is a typed connection between two resources. Additionally, a link is the primary means for a client to navigate from one resource to another.

A link is comprised of:

- o a link context,
- o a link relation type that identifies the semantics of the link (see Section 2.3.1),
- o a link target, identified by a URI, and
- o optionally, target attributes that further describe the link or the link target.

A link can be viewed as a statement of the form "{link context} has a {link relation type} resource at {link target}, which has {target attributes}" [6]. For example, the resource <http://example.com/> could have a "terms-of-service" resource at <http://example.com/tos>, which has a representation with the media type "text/html".

There are two special kinds of links:

- o An embedding link is a link with an additional hint: when the link is processed, it should be substituted with the representation of the referenced resource rather than cause the client to navigate away from the current resource. Thus, traversing an embedding link adds to the current state rather than replacing it.

The most well known example for an embedding link is the HTML element. When a Web browser processes this element, it automatically dereferences the "src" and renders the resulting image in place of the element.

- o A templated link is a link where the client constructs the link target URI from provided in-band instructions. The specific rules for such instructions are described by the representation format. URI Templates [3] provide a generic way to construct URIs through variable expansion.

Templated links allow a client to construct resource URIs without being coupled to the resource structure at the server, provided that the client learns the template from a representation sent by the server and does not have the template hard-coded.

2.3.1. Link Relation Types

A link relation type identifies the semantics of a link [6]. For example, a link with the relation type "copyright" indicates that the resource identified by the target URI is a statement of the copyright terms applying to the link context.

Relation types are not to be confused with media types; they do not identify the format of the representation that results when the link is dereferenced [6]. Rather, they only describe how the link context is related to another resource [6].

IANA maintains a list of registered link relation types at <http://www.iana.org/assignments/link-relations>.

Applications that don't wish to register a link relation type can use an extension link relation type [6]: a URI that uniquely identifies the link relation type. For example, an application can use the string "http://example.com/foo" as link relation type without having to register it. Using a URI to identify an extension link relation type, rather than a simple string, reduces the probability of different link relation types using the same identifiers.

2.3.2. Template Variable Names

A templated link enables clients to construct the target URI of a link, for example, when the link refers to a space of resources rather than a single resource. The most prominent mechanisms for this are URI Templates [3] and the HTML <form> element with a submission method of GET.

To enable an automated client to construct an URI reference from a URI Template, the name of the variable in the template can be used to identify the semantics of the variable. For example, when retrieving the representation of a collection of temperature readings, a variable named "threshold" could indicate the variable for setting a threshold of the readings to retrieve.

Template variable names are scoped to link relation types, i.e., two variables with the same name can have different semantics if they appear in links with different link relation types.

2.4. Forms

A form is the primary means for a client to submit information to a server, typically in order to change resource state.

A form is comprised of:

- o a form context,
- o a form relation type that identifies the semantics of the form (see Section 2.4.1),
- o a request method (e.g., PUT, POST, DELETE),
- o a submission URI,
- o a description of a representation that the server expects as part of the form submission, and
- o optionally, target attributes that further describe the form or the form target.

A form can be viewed as an instruction of the form "To perform a {form relation type} operation on {form context}, make a {request method} request to {submission URI}, which has {target attributes}". For example, to "update" the resource <http://example.com/config>, a client would make a PUT request to <http://example.com/config>. (In many cases, the target of a form is the same resource as the context, but this is not required.)

The description of the expected representation can be a set of form fields (see Section 2.4.2) or simply a list of acceptable media types.

Note: A form with a submission method of GET is, strictly speaking, a templated link, since it provides a way to construct a URI and does not submit a representation to the server.

2.4.1. Form Relation Types

A form relation type identifies the semantics of a form. For example, a form with the form relation type "create" indicates that a new item can be created within the form context by making a request to the resource identified by the target URI.

Similarly to extension link relation types, applications can use extension form relation types when they don't wish to register a form relation type.

2.4.2. Form Field Names

Forms can have a detailed description of the representation expected by the server as part of form submission. This description typically consists of a set of form fields where each form field is comprised

of a field name, a field type, and optionally a number of attributes such as a default value, a validation rule or a human-readable label.

To enable an automated client to fill out a form, the field name can be used to identify the semantics of the form field. For example, when controlling a smart light bulb, the field name "brightness" could indicate the field for setting the desired brightness of the light bulb.

Field names are scoped to form relation types, i.e., two form fields with the same name can have different semantics if they appear in forms with different form relation types.

The type of a form field is a data type such as "an integer between 1 and 100" or "an RGB color". The type is orthogonal to the field name, i.e., the type should not be determined from the field name even though the client can identify the semantics of the field from the name. This separation makes it easy to change the set of acceptable values in the future.

2.5. Well-Known Locations

Some applications may require the discovery of information about a host, known as "site-wide metadata" in RFC 5785 [4]. For example, RFC 6415 [19] defines a metadata document format for describing a host; similarly, RFC 6690 [22] defines a link format for the discovery of resources hosted by a server.

Applications that need to define a resource for this kind of metadata can register new "well-known locations". RFC 5785 [4] defines the path prefix `"/.well-known/"` in "http" and "https" URIs for this purpose. RFC 7252 [23] extends this convention to "coap" and "coaps" URIs.

IANA maintains a list of registered well-known URIs at <http://www.iana.org/assignments/well-known-uris>.

3. CoRE Application Descriptions

As applications are implemented and deployed, it becomes important to describe them in some structured way. This section provides a simple template for CoRE Application Descriptions. A uniform structure allows implementers to easily determine the components that make up the interface of an application.

The template below lists all components of applications that both the client and the server implementation of the application need to understand in order to interoperate. Crucially, items not listed in

the template are not part of the contract between clients and servers -- they are implementation details. This includes in particular the URIs of resources (see Section 4).

CoRE Application Descriptions are intended to be published in human-readable format by designers of applications and by operators of deployed application instances. Application designers may publish an application description as a general specification of all application instances, so that implementers can create interoperable clients and servers. Operators of application instances may publish an application description as part of the API documentation of the service, which should also include instructions how the service can be located and which communication protocols and security modes are used.

3.1. Template

The fields of the template are as follows:

Application name:

Name of the application. The name is not used to negotiate capabilities; it is purely informational. A name may include a version number or, for example, refer to a living standard that is updated continuously.

URI schemes:

URI schemes identifying the communication protocols that need to be understood by clients and servers. This information is mostly relevant for deployed instances of the application rather than for the general specification of the application.

Media types:

Internet media types that identify the representation formats that need to be understood by clients and servers. An application description must comprise at least one media type. Additional media types may be required or optional.

Link relation types:

Link relation types that identify the semantics of links. An application description may comprise IANA-registered link relation types and extension link relation types. Both may be required or optional.

Template variable names:

For each link relation type, variable names that identify the semantics of variables in templated links with that link relation type. Whether a template variable is required or optional is indicated in-band inside the templated link.

Form relation types:

Form relation types that identify the semantics of forms and, for each form relation type, the submission method(s) to be used. An application description may comprise IANA-registered form relation types and extension form relation types. Both may be required or optional.

Form field names:

For each form relation type, form field names that identify the semantics of form fields in forms with that form relation type. Whether a form field is required or optional is indicated in-band inside the form.

Well-known locations:

Well-known locations in the resource identifier space of servers that clients can use to discover information given the DNS name or IP address of a server.

Interoperability considerations:

Any issues regarding the interoperable use of the components of the application should be given here.

Security considerations:

Security considerations for the security of the application must be specified here.

Contact:

Person (including contact information) to contact for further information.

Author/Change controller:

Person (including contact information) authorized to change this application description.

Each field should include full citations for all specifications necessary to understand the application components.

4. URI Design Considerations

URIs [1] are a cornerstone of RESTful applications. They enable uniform identification of resources via URI schemes [7] and are used every time a client interacts with a particular resource or when a resource representation references 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 common for many RESTful applications to use these structures not only as an implementation detail but also make them part of the

public REST API, prescribing a fixed format for this data. However, there are a number of problems with this practice [5], in particular if the application designer and the server owner are not the same entity.

In hypermedia-driven applications, URIs are therefore not included in the application interface. A CoRE Application Description must not mandate any particular form of URI substructure.

RFC 7320 [5] describes the problematic practice of fixed URI structures in detail and provides some acceptable alternatives.

Nevertheless, the design of the URI structure on a server is an essential part of implementing a RESTful application, even though it is not part of the application interface. The server implementer is responsible for binding the resources identified by the application designer to URIs.

A good RESTful URI is:

- o Short. Short URIs are easier to remember and cause less overhead in requests and representations.
- o Meaningful. A URI should describe the resource in a way that is meaningful and useful to humans.
- o Consistent. URIs should follow a consistent pattern to make it easy to reason about the application.
- o Bookmarkable. Cool URIs don't change [9]. However, in practice, application resource structures do change. That should cause URIs to change as well so they better reflect reality. Implementations should not depend on unchanging URIs.
- o Shareable. A URI should not be context sensitive, e.g., to the currently logged-in user. It should be possible to share a URI with third parties so they can access the same resource.
- o Extension-less. Some applications return different data for different extensions, e.g., for "contacts.xml" or "contacts.json". But different URIs imply different resources. RESTful URIs should identify a single resource. Different representations of the resource can be negotiated (e.g., using the CoAP Accept option).

5. Security Considerations

The security considerations of RFC 3986 [1], RFC 5785 [4], RFC 6570 [3], RFC 6838 [2], RFC 7320 [5], RFC 7595 [7], and RFC 8288 [6] are inherited.

All components of an application description are expected to contain clear security considerations. CoRE Application Descriptions should furthermore contain security considerations that need to be taken into account for the security of the overall application.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

- [1] 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>>.
- [2] 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>>.
- [3] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [4] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [5] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.
- [6] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

- [7] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<https://www.rfc-editor.org/info/rfc7595>>.

7.2. Informative References

- [8] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [9] Berners-Lee, T., "Cool URIs don't change", 1998, <<http://www.w3.org/Provider/Style/URI.html>>.
- [10] 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>>.
- [11] 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>>.
- [12] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.
- [13] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., Yergeau, F., and J. Cowan, "Extensible Markup Language (XML) 1.1 (Second Edition)", World Wide Web Consortium Recommendation REC-xml11-20060816, August 2006, <<http://www.w3.org/TR/2006/REC-xml11-20060816>>.
- [14] 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>>.
- [15] 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>>.

- [16] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [17] Fielding, R., "REST APIs must be hypertext-driven", October 2008, <<http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>>.
- [18] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.
- [19] Hammer-Lahav, E., Ed. and B. Cook, "Web Host Metadata", RFC 6415, DOI 10.17487/RFC6415, October 2011, <<https://www.rfc-editor.org/info/rfc6415>>.
- [20] Masinter, L., "MIME and the Web", draft-masinter-mime-web-info-02 (work in progress), January 2011.
- [21] Richardson, L. and M. Amundsen, "RESTful Web APIs", O'Reilly Media, ISBN 978-1-4493-5806-8, September 2013.
- [22] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [23] 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>>.
- [24] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <<https://www.rfc-editor.org/info/rfc6906>>.

Acknowledgements

Jan Algermissen, Mike Amundsen, Mike Kelly, Julian Reschke, and Erik Wilde provided valuable input on link and form relation types.

Thanks to Olaf Bergmann, Carsten Bormann, Stefanie Gerdes, Ari Keranen, Michael Koster, Matthias Kovatsch, Teemu Savolainen, and Bilhanan Silverajan for helpful comments and discussions that have shaped the document.

Some of the text in this document has been borrowed from [5], [6], [17], and [20]. All errors are my own.

This work was funded in part by Nokia.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

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

K. Hartke
Ericsson
July 8, 2019

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

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 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 (<https://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	3
1.1. Notational Conventions	4
2. Data and Interaction Model	4
2.1. Browsing Context	4
2.2. Documents	5
2.3. Links	5
2.4. Forms	6
2.4.1. Form Fields	7
2.5. Embedded Representations	7
2.6. Navigation	7
2.7. History Traversal	9
3. Binary Format	9
3.1. Data Structure	10
3.1.1. Documents	10
3.1.2. Links	10
3.1.3. Forms	11
3.1.4. Embedded Representations	12
3.1.5. Directives	13
3.2. Dictionaries	13
3.2.1. Dictionary References	13
3.2.2. Media Type Parameter	14
4. Textual Format	14
4.1. Lexical Structure	15
4.1.1. Line Terminators	15
4.1.2. White Space	15
4.1.3. Comments	15
4.1.4. Identifiers	15
4.1.5. IRIs and IRI References	16
4.1.6. Literals	16
4.1.7. Punctuators	20
4.2. Syntactic Structure	20
4.2.1. Documents	20
4.2.2. Links	20
4.2.3. Forms	21
4.2.4. Embedded Representations	22
4.2.5. Directives	23
5. Usage Considerations	24
5.1. Specifying CoRAL-based Applications	24
5.1.1. Application Interfaces	24
5.1.2. Resource Names	25
5.1.3. Implementation Limits	25
5.2. Minting Vocabulary	26
5.3. Expressing Registered Link Relation Types	27
5.4. Expressing Simple RDF Statements	27
5.5. Expressing Language-Tagged Strings	27
5.6. Embedding CoRAL in CBOR Data	28

5.7. Submitting CoRAL Documents	28
5.7.1. PUT Requests	28
5.7.2. POST Requests	29
6. Security Considerations	29
7. IANA Considerations	30
7.1. Media Type "application/coral+cbor"	30
7.2. Media Type "text/coral"	32
7.3. CoAP Content Formats	33
7.4. CBOR Tag	33
8. References	34
8.1. Normative References	34
8.2. Informative References	36
Appendix A. Core Vocabulary	38
A.1. Base	38
A.2. Collections	39
A.3. HTTP	39
A.4. CoAP	40
Appendix B. Default Dictionary	41
Acknowledgements	41
Author's Address	42

1. Introduction

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

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

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

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

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

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

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

1.1. Notational Conventions

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

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

2. Data and Interaction Model

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

2.1. Browsing Context

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

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

2.2. Documents

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

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

2.3. Links

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

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

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

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

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

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

2.4. Forms

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

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

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

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

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

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

2.4.1. Form Fields

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

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

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

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

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

2.5. Embedded Representations

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

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

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

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

2.6. Navigation

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

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

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

1. The first step for the agent is to decide what to do next, i.e., which type of link to follow or form to submit, based on the link relation types and operation types it understands.
2. The agent then finds the link(s) or form(s) with the respective type in the active representation. This may yield one or more candidates, from which the agent will have to select the most appropriate one. The set of candidates may be empty, for example, when a transition is not supported or not allowed.
3. The agent selects one of the candidates based on the metadata associated with each of these. Metadata includes the content type of the target resource representation, the IRI scheme, the request method, and other information that is provided as nested elements in a link or form fields in a form.

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

4. The agent obtains the `_request IRI_` from the link target or submission target. Fragment identifiers are not part of the request IRI and MUST be separated from the rest of the IRI prior to a dereference.
5. The agent constructs a new request with the request IRI. If the agent is following a link, then the request method MUST be GET. If the agent is submitting a form, then the request method MUST be the one specified in the form. The request IRI may need to be converted to a URI (Section 3.1 of RFC 3987) for protocols that do not support IRIs.

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

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

6. The agent sends the request and receives the response.
7. If a fragment identifier was separated from the request IRI, the agent dereferences the fragment identifier within the received representation.
8. The agent `_updates the browsing context_` by making the (embedded or received) representation the active representation.
9. Finally, the agent processes the representation according to the semantics of the content type. If the representation is a CoRAL document (or any other representation that implements the CoRAL data and interaction model), this means the agent has the choice of what to do next again -- and the cycle repeats.

2.7. History Traversal

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

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

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

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

3. Binary Format

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

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

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

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

3.1. Data Structure

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

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

3.1.1. Documents

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

```
document = body
```

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

3.1.2. Links

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

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

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

```
relation-type = text
```

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

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

link-target = ciri / literal

ciri = <Defined in Section X of RFC XXXX>

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

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

3.1.3. Forms

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

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

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

operation-type = text

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

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

submission-target = ciri

3.1.3.1. Form Fields

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

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

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

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

form-field-type = text

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

form-field-value = ciri / literal

3.1.4. Embedded Representations

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

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

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

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

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

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

metadata-name = text

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

metadata-value = ciri / literal

3.1.5. Directives

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

directive = base-directive

3.1.5.1. Base Directives

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

base-directive = [1, base]

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

base = ciri

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

3.2. Dictionaries

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

3.2.1. Dictionary References

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

relation-type /= uint

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

operation-type /= uint

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

form-field-type /= uint

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

metadata-name /= uint

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

3.2.2. Media Type Parameter

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

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

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

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

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

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

4. Textual Format

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

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

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

4.1. Lexical Structure

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

token = identifier / iri / iriref / literal / punctuator

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

4.1.1. Line Terminators

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

4.1.2. White Space

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

4.1.3. Comments

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

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

4.1.4. Identifiers

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

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

START = <Any character with the XID_Start property>

CONTINUE = <Any character with the XID_Continue property>

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

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

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

4.1.1.5. IRIs and IRI References

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

iri = "<" IRI ">"

iriref = "<" IRI-reference ">"

IRI = <Defined in Section 2.2 of RFC 3987>

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

4.1.1.6. Literals

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

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

literal =/ null

4.1.1.6.1. Boolean Literals

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

boolean = "true" / "false"

4.1.6.2. Integer Literals

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

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

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

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

```
decimal = 1*DIGIT
```

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

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

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

```
DIGIT = %x30-39
```

```
BINDIG = %x30-31
```

```
OCTDIG = %x30-37
```

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

4.1.6.3. Floating-point Literals

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

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

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

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

```
fraction = "." 1*DIGIT
```

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

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

```
float =/ "NaN"
```

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

4.1.6.4. Date/Time Literals

Date/time literals denote an instant in time.

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

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

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

```
SQUOTE = %x27
```

4.1.6.5. Byte String Literals

Byte string literals denote an ordered sequence of bytes.

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

```
bytes = base16 / base32 / base64
```

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

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

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

4.1.6.6. Text String Literals

Text string literals denote a Unicode string.

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

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

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

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

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

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

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

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

`DQUOTE = %x22`

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

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

Table 1: Simple Escape Sequences

4.1.6.7. Null Literal

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

null = "null" / "_"

4.1.7. Punctuators

Punctuator tokens are used for grouping and separating.

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

4.2. Syntactic Structure

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

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

4.2.1. Documents

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

document = body

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

4.2.2. Links

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

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

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

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

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

simple-name = identifier

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

qualified-name = identifier ":" identifier

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

link-target = iriref / literal

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

4.2.3. Forms

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

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

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

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

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

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

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

submission-target = iriref

4.2.3.1. Form Fields

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

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

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

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

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

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

form-field-value = iriref / literal

4.2.4. Embedded Representations

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

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

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

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

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

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

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

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

metadata-value = iriref / literal

4.2.5. Directives

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

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

directive = base-directive / using-directive

4.2.5.1. Base Directives

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

base-directive = "#" "base" base

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

base = iriref

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

4.2.5.2. Using Directives

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


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

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

5. Usage Considerations

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

5.1. Specifying CoRAL-based Applications

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

- o Resources are identified by IRIs or represented by value literals.
- o 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.
- o Representations are CoRAL documents encoded in the binary format defined in Section 3 or the textual format defined in Section 4. Depending on the application, additional representation formats may be used.

5.1.1. Application Interfaces

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

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

- o Form field types that identify the semantics of form fields. Additionally, for each form field type, the permissible form field values.
- o Metadata names that identify the semantics of representation metadata. Additionally, for each metadata name, the permissible metadata values.

5.1.2. Resource Names

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

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

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

5.1.3. Implementation Limits

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

Applications may also mandate the following and other restrictions:

- o use of only either the binary format or the text format;
- o use of only either HTTP or CoAP as supported Web transfer protocol;
- o use of only dictionary references in the binary format for certain vocabulary;

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

5.2. Minting Vocabulary

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

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

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

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

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

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

5.3. Expressing Registered Link Relation Types

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

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

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

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

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

5.4. Expressing Simple RDF Statements

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

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

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

5.5. Expressing Language-Tagged Strings

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

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

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

5.6. Embedding CoRAL in CBOR Data

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

CoRAL-Document = document

CoRAL-Link = link

CoRAL-Form = form

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

5.7. Submitting CoRAL Documents

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

5.7.1. PUT Requests

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

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

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

5.7.2. POST Requests

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

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

6. Security Considerations

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

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

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

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

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

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

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

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

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

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

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

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

7. IANA Considerations

7.1. Media Type "application/coral+cbor"

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

Type name:
application

Subtype name:
coral+cbor

Required parameters:

N/A

Optional parameters:

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

Encoding considerations:

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

Security considerations:

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

Interoperability considerations:

N/A

Published specification:

[I-D.hartke-t2trg-coral]

Applications that use this media type:

See Section 1 of [I-D.hartke-t2trg-coral].

Fragment identifier considerations:

As specified for "application/cbor".

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): .coral.cbor

Macintosh file type code(s): N/A

Person & email address to contact for further information:

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

Intended usage:

COMMON

Restrictions on usage:

N/A

Author:

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

Change controller:

IESG

Provisional registration?

No

7.2. Media Type "text/coral"

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

Type name:
text

Subtype name:
coral

Required parameters:
N/A

Optional parameters:
N/A

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

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

Interoperability considerations:
N/A

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

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

Fragment identifier considerations:
N/A

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

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

Intended usage:
COMMON

Restrictions on usage:

N/A

Author:

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

Change controller:

IESG

Provisional registration?

No

7.3. CoAP Content Formats

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

- o Content Type: application/coral+cbor
Content Coding: identity
ID: TBD3
Reference: [I-D.hartke-t2trg-coral]
- o Content Type: text/coral
Content Coding: identity
ID: TBD4
Reference: [I-D.hartke-t2trg-coral]

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

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

7.4. CBOR Tag

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

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

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

8. References

8.1. Normative References

- [I-D.hartke-t2trg-ciri]
Hartke, K., "Constrained Internationalized Resource Identifiers", draft-hartke-t2trg-ciri-03 (work in progress), July 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <<https://www.rfc-editor.org/info/rfc3987>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.

- [RFC6657] Melnikov, A. and J. Reschke, "Update to MIME regarding "charset" Parameter Handling in Textual Media Types", RFC 6657, DOI 10.17487/RFC6657, July 2012, <<https://www.rfc-editor.org/info/rfc6657>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <<https://www.rfc-editor.org/info/rfc6943>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [UNICODE] The Unicode Consortium, "The Unicode Standard", <<http://www.unicode.org/versions/latest/>>.
- Note that this reference is to the latest version of Unicode, rather than to a specific release. It is not expected that future changes in the Unicode specification will have any impact on this document.
- [UNICODE-UAX15] The Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", <<http://unicode.org/reports/tr15/>>.
- [UNICODE-UAX31] The Unicode Consortium, "Unicode Standard Annex #31: Unicode Identifier and Pattern Syntax", <<http://unicode.org/reports/tr31/>>.

[UNICODE-UAX36]

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

8.2. Informative References

[CAPEC-197]

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

[CAPEC-201]

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

[FOAF]

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

[RFC4287]

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

[RFC5789]

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

[RFC6573]

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

[RFC6690]

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

[RFC7228]

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

[RFC7230]

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

- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [W3C.REC-html52-20171214]
Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, December 2017, <<https://www.w3.org/TR/2017/REC-html52-20171214>>.
- [W3C.REC-rdf-schema-20140225]
Brickley, D. and R. Guha, "RDF Schema 1.1", World Wide Web Consortium Recommendation REC-rdf-schema-20140225, February 2014, <<http://www.w3.org/TR/2014/REC-rdf-schema-20140225>>.
- [W3C.REC-rdf11-concepts-20140225]
Cyganiak, R., Wood, D., and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax", World Wide Web Consortium Recommendation REC-rdf11-concepts-20140225, February 2014, <<http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225>>.

[W3C.REC-turtle-20140225]

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

[W3C.REC-webarch-20041215]

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

Appendix A. Core Vocabulary

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

A.1. Base

Link Relation Types:

<<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>>

Indicates that the link's context is an instance of the class specified as the link's target, as defined by RDF Schema [W3C.REC-rdf-schema-20140225].

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

Indicates that the link target is a language tag [RFC5646] that specifies the language of the link context.

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

Operation Types:

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

Indicates that the state of the form's context can be replaced with the state described by a representation submitted to the server.

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

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

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

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

A.2. Collections

Link Relation Types:

<http://www.iana.org/assignments/relation/item>

Indicates that the link's context is a collection and that the link's target is a member of that collection, as defined in Section 2.1 of RFC 6573 [RFC6573].

<http://www.iana.org/assignments/relation/collection>

Indicates that the link's target is a collection and that the link's context is a member of that collection, as defined in Section 2.2 of RFC 6573 [RFC6573].

Operation Types:

<http://coreapps.org/collections#create>

Indicates that the form's context is a collection and that a new item can be created in that collection with the state defined by a representation submitted to the server.

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

<http://coreapps.org/collections#delete>

Indicates that the form's context is a member of a collection and that the form's context can be removed from that collection.

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

A.3. HTTP

Form Field Types:

<http://coreapps.org/http#method>

Specifies the HTTP method for the request.

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

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

<http://coreapps.org/http#accept>

Specifies an acceptable HTTP content type for the request payload. There may be multiple form fields of this type. If a form does not include a form field of this type, the server accepts any or no request payload, depending on the operation type.

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

Representation Metadata:

<http://coreapps.org/http#type>

Specifies the HTTP content type of the representation.

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

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

A.4. CoAP

Form Field Types:

<http://coreapps.org/coap#method>

Specifies the CoAP method for the request.

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

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

<http://coreapps.org/coap#accept>

Specifies an acceptable CoAP content format for the request payload. There may be multiple form fields of this type. If a form does not include a form field of this type, the server

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

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

Representation Metadata:

<http://coreapps.org/coap#type>

Specifies the CoAP content format of the representation.

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

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

Appendix B. Default Dictionary

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

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

Table 2: Default Dictionary

Acknowledgements

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

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

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 18, 2018

A. Keranen
Ericsson
M. Kovatsch
ETH Zurich
K. Hartke
Universitaet Bremen TZI
September 14, 2017

RESTful Design for Internet of Things Systems
draft-keranen-t2trg-rest-iot-05

Abstract

This document gives guidance for designing Internet of Things (IoT) systems that follow the principles of the Representational State Transfer (REST) architectural style.

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 March 18, 2018.

Copyright Notice

Copyright (c) 2017 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	3
2. Terminology	3
3. Basics	6
3.1. Architecture	6
3.2. System design	8
3.3. Uniform Resource Identifiers (URIs)	9
3.4. Representations	10
3.5. HTTP/CoAP Methods	10
3.5.1. GET	11
3.5.2. POST	11
3.5.3. PUT	12
3.5.4. DELETE	12
3.6. HTTP/CoAP Status/Response Codes	12
4. REST Constraints	13
4.1. Client-Server	13
4.2. Stateless	14
4.3. Cache	14
4.4. Uniform Interface	14
4.5. Layered System	15
4.6. Code-on-Demand	15
5. Hypermedia-driven Applications	16
5.1. Motivation	16
5.2. Knowledge	17
5.3. Interaction	18
6. Design Patterns	18
6.1. Collections	18
6.2. Calling a Procedure	19
6.2.1. Instantly Returning Procedures	19
6.2.2. Long-running Procedures	19
6.2.3. Conversion	20
6.2.4. Events as State	20
6.3. Server Push	21
7. Security Considerations	22
8. Acknowledgement	23
9. References	23
9.1. Normative References	23
9.2. Informative References	25
Appendix A. Future Work	26
Authors' Addresses	26

1. Introduction

The Representational State Transfer (REST) architectural style [REST] is a set of guidelines and best practices for building distributed hypermedia systems. At its core is a set of constraints, which when fulfilled enable desirable properties for distributed software systems such as scalability and modifiability. When REST principles are applied to the design of a system, the result is often called RESTful and in particular an API following these principles is called a RESTful API.

Different protocols can be used with RESTful systems, but at the time of writing the most common protocols are HTTP [RFC7230] and CoAP [RFC7252]. Since RESTful APIs are often simple and lightweight, they are a good fit for various IoT applications. The goal of this document is to give basic guidance for designing RESTful systems and APIs for IoT applications and give pointers for more information. Design of a good RESTful IoT system has naturally many commonalities with other Web systems. Compared to other systems, the key characteristics of many IoT systems include:

- o data formats, interaction patterns, and other mechanisms that minimize, or preferably avoid, the need for human interaction
- o preference for compact and simple data formats to facilitate efficient transfer over (often) constrained networks and lightweight processing in constrained nodes

2. Terminology

This section explains some of the common terminology that is used in the context of RESTful design for IoT systems. For terminology of constrained nodes and networks, see [RFC7228].

Cache: A local store of response messages and the subsystem that controls storage, retrieval, and deletion of messages in it.

Client: A node that sends requests to servers and receives responses. In RESTful IoT systems it's common for nodes to have more than one role (e.g., both server and client; see Section 3.1).

Client State: The state kept by a client between requests. This typically includes the currently processed representation, the set of active requests, the history of requests, bookmarks (URIs stored for later retrieval), and application-specific state (e.g., local variables). (Note that this is called "Application State" in [REST], which has some ambiguity in modern (IoT) systems where

the overall state of the distributed application (i.e., application state) is reflected in the union of all Client States and Resource States of all clients and servers involved.)

Content Negotiation: The practice of determining the "best" representation for a client when examining the current state of a resource. The most common forms of content negotiation are Proactive Content Negotiation and Reactive Content Negotiation.

Form: A hypermedia control that enables a client to change the state of a resource or to construct a query locally.

Forward Proxy: An intermediary that is selected by a client, usually via local configuration rules, and that can be tasked to make requests on behalf of the client. This may be useful, for example, when the client lacks the capability to make the request itself or to service the response from a cache in order to reduce response time, network bandwidth, and energy consumption.

Gateway: A reverse proxy that provides an interface to a non-RESTful system such as legacy systems or alternative technologies such as Bluetooth ATT/GATT. See also "Reverse Proxy".

Hypermedia Control: A component, such as a link or a form, embedded in a representation that identifies a resource for future hypermedia interactions. If the client engages in an interaction with the identified resource, the result may be a change to resource state and/or client state.

Idempotent Method: A method where multiple identical requests with that method lead to the same visible resource state as a single such request.

Link: A hypermedia control that enables a client to navigate between resources and thereby change the client state.

Link Relation Type: An identifier that describes how the link target resource relates to the current resource (see [RFC5988]).

Media Type: A string such as "text/html" or "application/json" that is used to label representations so that it is known how the representation should be interpreted and how it is encoded.

Method: An operation associated with a resource. Common methods include GET, PUT, POST, and DELETE (see Section 3.5 for details).

Origin Server: A server that is the definitive source for representations of its resources and the ultimate recipient of any

request that intends to modify its resources. In contrast, intermediaries (such as proxies caching a representation) can assume the role of a server, but are not the source for representations as these are acquired from the origin server.

Proactive Content Negotiation: A content negotiation mechanism where the server selects a representation based on the expressed preference of the client. For example, an IoT application could send a request to a sensor with preferred media type "application/senml+json".

Reactive Content Negotiation: A content negotiation mechanism where the client selects a representation from a list of available representations. The list may, for example, be included by a server in an initial response. If the user agent is not satisfied by the initial response representation, it can request one or more of the alternative representations, selected based on metadata (e.g., available media types) included in the response.

Representation: A serialization that represents the current or intended state of a resource and that can be transferred between clients and servers. REST requires representations to be self-describing, meaning that there must be metadata that allows peers to understand which representation format is used. Depending on the protocol needs and capabilities, there can be additional metadata that is transmitted along with the representation.

Representation Format: A set of rules for serializing resource state. On the Web, the most prevalent representation format is HTML. Other common formats include plain text and formats based on JSON [RFC7159], XML, or RDF. Within IoT systems, often compact formats based on JSON, CBOR [RFC7049], and EXI [W3C.REC-exi-20110310] are used.

Representational State Transfer (REST): An architectural style for Internet-scale distributed hypermedia systems.

Resource: An item of interest identified by a URI. Anything that can be named can be a resource. A resource often encapsulates a piece of state in a system. Typical resources in an IoT system can be, e.g., a sensor, the current value of a sensor, the location of a device, or the current state of an actuator.

Resource State: A model of a resource's possible states that is represented in a supported representation type, typically a media type. Resources can change state because of REST interactions with them, or they can change state for reasons outside of the REST model.

Resource Type: An identifier that annotates the application-
semantics of a resource (see Section 3.1 of [RFC6690]).

Reverse Proxy: An intermediary that appears as a server towards the
client but satisfies the requests by forwarding them to the actual
server (possibly via one or more other intermediaries). A reverse
proxy is often used to encapsulate legacy services, to improve
server performance through caching, and to enable load balancing
across multiple machines.

Safe Method: A method that does not result in any state change on
the origin server when applied to a resource.

Server: A node that listens for requests, performs the requested
operation and sends responses back to the clients.

Uniform Resource Identifier (URI): A global identifier for
resources. See Section 3.3 for more details.

3. Basics

3.1. Architecture

The components of a RESTful system are assigned one or both of two
roles: client or server. Note that the terms "client" and "server"
refer only to the roles that the nodes assume for a particular
message exchange. The same node might act as a client in some
communications and a server in others. Classic user agents (e.g.,
Web browsers) are always in the client role and have the initiative
to issue requests. Origin servers always have the server role and
govern over the resources they host.

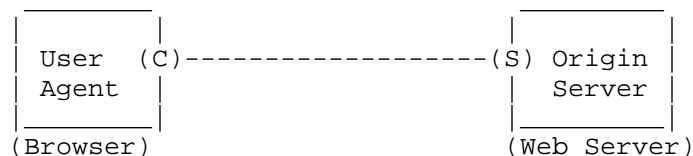


Figure 1: Client-Server Communication

Intermediaries (such as forward proxies, reverse proxies, and
gateways) implement both roles, but only forward requests to other
intermediaries or origin servers. They can also translate requests
to different protocols, for instance, as CoAP-HTTP cross-proxies.

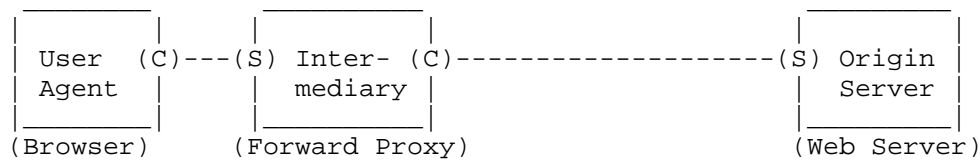


Figure 2: Communication with Forward Proxy

Reverse proxies are usually imposed by the origin server. In addition to the features of a forward proxy, they can also provide an interface for non-RESTful services such as legacy systems or alternative technologies such as Bluetooth ATT/GATT. In this case, reverse proxies are usually called gateways. This property is enabled by the Layered System constraint of REST, which says that a client cannot see beyond the server it is connected to (i.e., it is left unaware of the protocol/paradigm change).

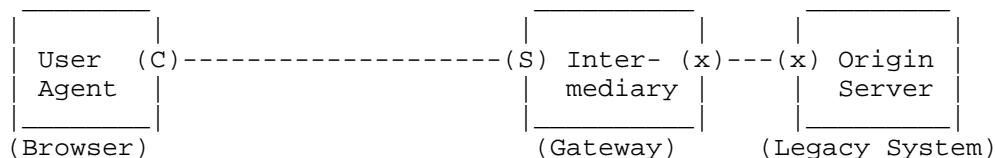


Figure 3: Communication with Reverse Proxy

Nodes in IoT systems often implement both roles. Unlike intermediaries, however, they can take the initiative as a client (e.g., to register with a directory, such as CoRE Resource Directory [I-D.ietf-core-resource-directory], or to interact with another thing) and act as origin server at the same time (e.g., to serve sensor values or provide an actuator interface).

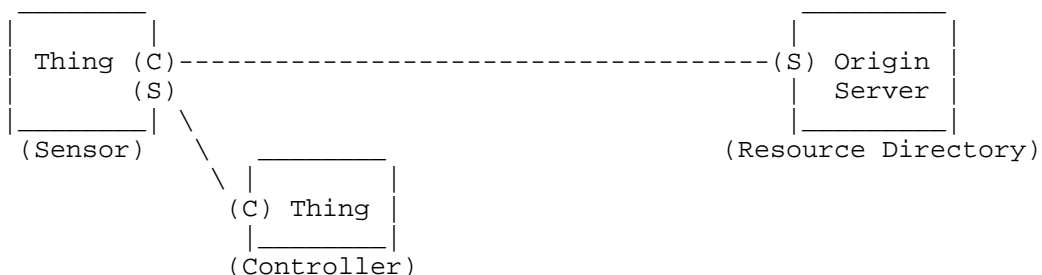


Figure 4: Constrained RESTful environments

3.2. System design

When designing a RESTful system, the primary effort goes into modeling the state of the distributed application and assigning it to the different components (i.e., clients and servers). How clients can navigate through the resources and modify state to achieve their goals is defined through hypermedia controls, that is, links and forms. Hypermedia controls span a kind of a state machine where the nodes are resources and the transitions are links or forms. Clients run this state machine (i.e., the application) by retrieving representations, processing the data, and following the included hypermedia controls. In REST, remote state is changed by submitting forms. This is usually done by retrieving the current state, modifying the state on the client side, and transferring the new state to the server in the form of new representations - rather than calling a service and modifying the state on the server side.

Client state encompasses the current state of the described state machine and the possible next transitions derived from the hypermedia controls within the currently processed representation (see Section 2). Furthermore, clients can have part of the state of the distributed application in local variables.

Resource state includes the more persistent data of an application (i.e., independent of individual clients). This can be static data such as device descriptions, persistent data such as system configurations, but also dynamic data such as the current value of a sensor on a thing.

It is important to distinguish between "client state" and "resource state" and keep them separate. Following the Stateless constraint, the client state must be kept only on clients. That is, there is no establishment of shared information about past and future interactions between client and server (usually called a session). On the one hand, this makes requests a bit more verbose since every request must contain all the information necessary to process it. On the other hand, this makes servers efficient and scalable, since they do not have to keep any state about their clients. Requests can easily be distributed over multiple worker threads or server instances. For IoT systems, this constraint lowers the memory requirements for server implementations, which is particularly important for constrained servers (e.g., sensor nodes) and servers serving large amount of clients (e.g., Resource Directory).

3.3. Uniform Resource Identifiers (URIs)

An important part of RESTful API design is to model the system as a set of resources whose state can be retrieved and/or modified and where resources can be potentially also created and/or deleted.

Uniform Resource Identifiers (URIs) are used to indicate a resource for interaction, to reference a resource from another resource, to advertise or bookmark a resource, or to index a resource by search engines.

```
foo://example.com:8042/over/there?name=ferret#nose
  \_/      \_____/ \_____/ \_____/ \_____/
  |         |         |         |         |
scheme authority  path    query  fragment
```

A URI is a sequence of characters that matches the syntax defined in [RFC3986]. It consists of a hierarchical sequence of five components: scheme, authority, path, query, and fragment (from most significant to least significant). A scheme creates a namespace for resources and defines how the following components identify a resource within that namespace. The authority identifies an entity that governs part of the namespace, such as the server "www.example.org" in the "http" scheme. A host name (e.g., a fully qualified domain name) or an IP address, potentially followed by a transport layer port number, are usually used in the authority component for the "http" and "coap" schemes. The path and query contain data to identify a resource within the scope of the URI's scheme and naming authority. The fragment allows to refer to some portion of the resource, such as a Record in a SenML Pack. However, fragments are processed only at client side and not sent on the wire. [RFC7320] provides more details on URI design and ownership with best current practices for establishing URI structures, conventions, and formats.

For RESTful IoT applications, typical schemes include "https", "coaps", "http", and "coap". These refer to HTTP and CoAP, with and without Transport Layer Security (TLS) [RFC5246]. (CoAP uses Datagram TLS (DTLS) [RFC6347], the variant of TLS for UDP.) These four schemes also provide means for locating the resource; using the HTTP protocol for "http" and "https", and with the CoAP protocol for "coap" and "coaps". If the scheme is different for two URIs (e.g., "coap" vs. "coaps"), it is important to note that even if the rest of the URI is identical, these are two different resources, in two distinct namespaces.

The query parameters can be used to parametrize the resource. For example, a GET request may use query parameters to request the server

to send only certain kind data of the resource (i.e., filtering the response). Query parameters in PUT and POST requests do not have such established semantics and are not commonly used. Whether the order of the query parameters matters in URIs is unspecified and they can be re-ordered e.g., by proxies. Therefore applications should not rely on their order; see Section 3.3 of [RFC6943] for more details.

3.4. Representations

Clients can retrieve the resource state from an origin server or manipulate resource state on the origin server by transferring resource representations. Resource representations have a media type that tells how the representation should be interpreted by identifying the representation format used.

Typical media types for IoT systems include:

- o "text/plain" for simple UTF-8 text
- o "application/octet-stream" for arbitrary binary data
- o "application/json" for the JSON format [RFC7159]
- o "application/senml+json" [I-D.ietf-core-senml] for Sensor Markup Language (SenML) formatted data
- o "application/cbor" for CBOR [RFC7049]
- o "application/exi" for EXI [W3C.REC-exi-20110310]

A full list of registered Internet Media Types is available at the IANA registry [IANA-media-types] and numerical media types registered for use with CoAP are listed at CoAP Content-Formats IANA registry [IANA-CoAP-media].

3.5. HTTP/CoAP Methods

Section 4.3 of [RFC7231] defines the set of methods in HTTP; Section 5.8 of [RFC7252] defines the set of methods in CoAP. As part of the Uniform Interface constraint, each method can have certain properties that give guarantees to clients.

Safe methods do not cause any state change on the origin server when applied to a resource. For example, the GET method only returns a representation of the resource state but does not change the resource. Thus, it is always safe for a client to retrieve a representation without affecting server-side state.

Idempotent methods can be applied multiple times to the same resource while causing the same visible resource state as a single such request. For example, the PUT method replaces the state of a resource with a new state; replacing the state multiple times with the same new state still results in the same state for the resource. However, the response from the server can be different when the same idempotent method is used multiple times. For example when DELETE is used twice on an existing resource, the first request would remove the association and return success acknowledgement whereas the second request would likely result in error response due to non-existing resource.

The following lists the most relevant methods and gives a short explanation of their semantics.

3.5.1. GET

The GET method requests a current representation for the target resource, while the origin server must ensure that there are no side-effects on the resource state. Only the origin server needs to know how each of its resource identifiers corresponds to an implementation and how each implementation manages to select and send a current representation of the target resource in a response to GET.

A payload within a GET request message has no defined semantics.

The GET method is safe and idempotent.

3.5.2. POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics.

If one or more resources has been created on the origin server as a result of successfully processing a POST request, the origin server sends a 201 (Created) response containing a Location header field (with HTTP) or Location-Path and/or Location-Query Options (with CoAP) that provide an identifier for the resource created. The server also includes a representation that describes the status of the request while referring to the new resource(s).

The POST method is not safe nor idempotent.

3.5.3. PUT

The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being sent.

The fundamental difference between the POST and PUT methods is highlighted by the different intent for the enclosed representation. The target resource in a POST request is intended to handle the enclosed representation according to the resource's own semantics, whereas the enclosed representation in a PUT request is defined as replacing the state of the target resource. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

The PUT method is not safe, but is idempotent.

3.5.4. DELETE

The DELETE method requests that the origin server remove the association between the target resource and its current functionality.

If the target resource has one or more current representations, they might or might not be destroyed by the origin server, and the associated storage might or might not be reclaimed, depending entirely on the nature of the resource and its implementation by the origin server.

The DELETE method is not safe, but is idempotent.

3.6. HTTP/CoAP Status/Response Codes

Section 6 of [RFC7231] defines a set of Status Codes in HTTP that are used by application to indicate whether a request was understood and satisfied, and how to interpret the answer. Similarly, Section 5.9 of [RFC7252] defines the set of Response Codes in CoAP.

The status codes consist of three digits (e.g., "404" with HTTP or "4.04" with CoAP) where the first digit expresses the class of the code. Implementations do not need to understand all status codes, but the class of the code must be understood. Codes starting with 1 are informational; the request was received and being processed. Codes starting with 2 indicate a successful request. Codes starting with 3 indicate redirection; further action is needed to complete the

request. Codes starting with 4 and 5 indicate errors. The codes starting with 4 mean client error (e.g., bad syntax in the request) whereas codes starting with 5 mean server error; there was no apparent problem with the request, but server was not able to fulfill the request.

Responses may be stored in a cache to satisfy future, equivalent requests. HTTP and CoAP use two different patterns to decide what responses are cacheable. In HTTP, the cacheability of a response depends on the request method (e.g., responses returned in reply to a GET request are cacheable). In CoAP, the cacheability of a response depends on the response code (e.g., responses with code 2.04 are cacheable). This difference also leads to slightly different semantics for the codes starting with 2; for example, CoAP does not have a 2.00 response code whereas 200 ("OK") is commonly used with HTTP.

4. REST Constraints

The REST architectural style defines a set of constraints for the system design. When all constraints are applied correctly, REST enables architectural properties of key interest [REST]:

- o Performance
- o Scalability
- o Reliability
- o Simplicity
- o Modifiability
- o Visibility
- o Portability

The following sub-sections briefly summarize the REST constraints and explain how they enable the listed properties.

4.1. Client-Server

As explained in the Architecture section, RESTful system components have clear roles in every interaction. Clients have the initiative to issue requests, intermediaries can only forward requests, and servers respond requests, while origin servers are the ultimate recipient of requests that intent to modify resource state.

This improves simplicity and visibility, as it is clear which component started an interaction. Furthermore, it improves modifiability through a clear separation of concerns.

4.2. Stateless

The Stateless constraint requires messages to be self-contained. They must contain all the information to process it, independent from previous messages. This allows to strictly separate the client state from the resource state.

This improves scalability and reliability, since servers or worker threads can be replicated. It also improves visibility because message traces contain all the information to understand the logged interactions.

Furthermore, the Stateless constraint enables caching.

4.3. Cache

This constraint requires responses to have implicit or explicit cache-control metadata. This enables clients and intermediary to store responses and re-use them to locally answer future requests. The cache-control metadata is necessary to decide whether the information in the cached response is still fresh or stale and needs to be discarded.

Cache improves performance, as less data needs to be transferred and response times can be reduced significantly. Less transfers also improves scalability, as origin servers can be protected from too many requests. Local caches furthermore improve reliability, since requests can be answered even if the origin server is temporarily not available.

4.4. Uniform Interface

All RESTful APIs use the same, uniform interface independent of the application. This simple interaction model is enabled by exchanging representations and modifying state locally, which simplifies the interface between clients and servers to a small set of methods to retrieve, update, and delete state - which applies to all applications.

In contrast, in a service-oriented RPC approach, all required ways to modify state need to be modeled explicitly in the interface resulting in a large set of methods - which differs from application to application. Moreover, it is also likely that different parties come up with different ways how to modify state, including the naming of

the procedures, while the state within an application is a bit easier to agree on.

A REST interface is fully defined by:

- o URIs to identify resources
- o representation formats to represent (and retrieve and manipulate) resource state
- o self-descriptive messages with a standard set of methods (e.g., GET, POST, PUT, DELETE with their guaranteed properties)
- o hypermedia controls within representations

The concept of hypermedia controls is also known as HATEOAS: Hypermedia As The Engine Of Application State. The origin server embeds controls for the interface into its representations and thereby informs the client about possible next requests. The mostly used control for RESTful systems is Web Linking [RFC5590]. Hypermedia forms are more powerful controls that describe how to construct more complex requests, including representations to modify resource state.

While this is the most complex constraints (in particular the hypermedia controls), it improves many different key properties. It improves simplicity, as uniform interfaces are easier to understand. The self-descriptive messages improve visibility. The limitation to a known set of representation formats fosters portability. Most of all, however, this constraint is the key to modifiability, as hypermedia-driven, uniform interfaces allow clients and servers to evolve independently, and hence enable a system to evolve.

4.5. Layered System

This constraint enforces that a client cannot see beyond the server with which it is interacting.

A layered system is easier to modify, as topology changes become transparent. Furthermore, this helps scalability, as intermediaries such as load balancers can be introduced without changing the client side. The clean separation of concerns helps with simplicity.

4.6. Code-on-Demand

This principle enables origin servers to ship code to clients.

Code-on-Demand improves modifiability, since new features can be deployed during runtime (e.g., support for a new representation format). It also improves performance, as the server can provide code for local pre-processing before transferring the data.

5. Hypermedia-driven Applications

Hypermedia-driven applications take advantage of hypermedia controls, i.e., links and forms, embedded in the resource representations. A hypermedia client is a client that is capable of processing these hypermedia controls. Hypermedia links can be used to give additional information about a resource representation (e.g., the source URI of the representation) or pointing to other resources. The forms can be used to describe the structure of the data that can be sent (e.g., with a POST or PUT method) to a server, or how a data retrieval (e.g., GET) request for a resource should be formed. In a hypermedia-driven application the client interacts with the server using only the hypermedia controls, instead of selecting methods and/or constructing URIs based on out-of-band information, such as API documentation.

5.1. Motivation

The advantage of this approach is increased evolvability and extensibility. This is important in scenarios where servers exhibit a range of feature variations, where it's expensive to keep evolving client knowledge and server knowledge in sync all the time, or where there are many different client and server implementations. Hypermedia controls serve as indicators in capability negotiation. In particular, they describe available resources and possible operations on these resources using links and forms, respectively.

There are multiple reasons why a server might introduce new links or forms:

- o The server implements a newer version of the application. Older clients ignore the new links and forms, while newer clients are able to take advantage of the new features by following the new links and submitting the new forms.
- o The server offers links and forms depending on the current state. The server can tell the client which operations are currently valid and thus help the client navigate the application state machine. The client does not have to have knowledge which operations are allowed in the current state or make a request just to find out that the operation is not valid.

- o The server offers links and forms depending on the client's access control rights. If the client is unauthorized to perform a certain operation, then the server can simply omit the links and forms for that operation.

5.2. Knowledge

A client needs to have knowledge of a couple of things for successful interaction with a server. This includes what resources are available, what representations of resource states are available, what each representation describes, how to retrieve a representation, what state changing operations on a resource are possible, how to perform these operations, and so on.

Some part of this knowledge, such as how to retrieve the representation of a resource state, is typically hard-coded in the client software. For other parts, a choice can often be made between hard-coding the knowledge or acquiring it on-demand. The key to success in either case is the use in-band information for identifying the knowledge that is required. This enables the client to verify that it has all required knowledge and to acquire missing knowledge on-demand.

A hypermedia-driven application typically uses the following identifiers:

- o URI schemes that identify communication protocols,
- o Internet Media Types that identify representation formats,
- o link relation types or resource types that identify link semantics,
- o form relation types that identify form semantics,
- o variable names that identify the semantics of variables in templated links, and
- o form field names that identify the semantics of form fields in forms.

The knowledge about these identifiers as well as matching implementations have to be shared a priori in a RESTful system.

5.3. Interaction

A client begins interacting with an application through a GET request on an entry point URI. The entry point URI is the only URI a client is expected to know before interacting with an application. From there, the client is expected to make all requests by following links and submitting forms that are provided in previous responses. The entry point URI can be obtained, for example, by manual configuration or some discovery process (e.g., DNS-SD [RFC6763] or Resource Directory [I-D.ietf-core-resource-directory]). For Constrained RESTful environments `"/.well-known/core"` relative URI is defined as a default entry point for requesting the links hosted by servers with known or discovered addresses [RFC6690].

6. Design Patterns

Certain kinds of design problems are often recurring in variety of domains, and often re-usable design patterns can be applied to them. Also some interactions with a RESTful IoT system are straightforward to design; a classic example of reading a temperature from a thermometer device is almost always implemented as a GET request to a resource that represents the current value of the thermometer. However, certain interactions, for example data conversions or event handling, do not have as straightforward and well established ways to represent the logic with resources and REST methods.

The following sections describe how common design problems such as different interactions can be modeled with REST and what are the benefits of different approaches.

6.1. Collections

A common pattern in RESTful systems across different domains is the collection. A collection can be used to combine multiple resources together by providing resources that consist of set of (often partial) representations of resources, called items, and links to resources. The collection resource also defines hypermedia controls for managing and searching the items in the collection.

Examples of the collection pattern in RESTful IoT systems are the CoRE Resource Directory [I-D.ietf-core-resource-directory], CoAP pub/sub broker [I-D.ietf-core-coap-pubsub], and resource discovery via `"/.well-known/core"`. Collection+JSON [CollectionJSON] is an example of a generic collection Media Type.

6.2. Calling a Procedure

To modify resource state, clients usually use GET to retrieve a representation from the server, modify that locally, and transfer the resulting state back to the server with a PUT (see Section 4.4). Sometimes, however, the state can only be modified on the server side, for instance, because representations would be too large to transfer or part of the required information shall not be accessible to clients. In this case, resource state is modified by calling a procedure (or "function"). This is usually modeled with a POST request, as this method leaves the behavior semantics completely to the server. Procedure calls can be divided into two different classes based on how long they are expected to execute: "instantly" returning and long-running.

6.2.1. Instantly Returning Procedures

When the procedure can return within the expected response time of the system, the result can be directly returned in the response. The result can either be actual content or just a confirmation that the call was successful. In either case, the response does not contain a representation of the resource, but a so-called action result. Action results can still have hypermedia controls to provide the possible transitions in the application state machine.

6.2.2. Long-running Procedures

When the procedure takes longer than the expected response time of the system, or even longer than the response timeout, it is a good pattern to create a new resource to track the "task" execution. The server would respond instantly with a "Created" status (HTTP code 201 or CoAP 2.01) and indicate the location of the task resource in the corresponding header field (or CoAP option) or as a link in the action result. The created resource can be used to monitor the progress, to potentially modify queued tasks or cancel tasks, and to eventually retrieve the result.

Monitoring information would be modeled as state of the task resource, and hence be retrievable as representation. The result - when available - can be embedded in the representation or given as a link to another sub-resource. Modifying tasks can be modeled with forms that either update sub-resources via PUT or do a partial write using PATCH or POST. Canceling a task would be modeled with a form that uses DELETE to remove the task resource.

6.2.3. Conversion

A conversion service is a good example where REST resources need to behave more like a procedure call. The knowledge of converting from one representation to another is located only at the server to relieve clients from high processing or storing lots of data. There are different approaches that all depend on the particular conversion problem.

As mentioned in the previous sections, POST request are a good way to model functionality that does not necessarily affect resource state. When the input data for the conversion is small and the conversion result is deterministic, however, it can be better to use a GET request with the input data in the URI query part. The query is parameterizing the conversion resource, so that it acts like a look-up table. The benefit is that results can be cached also for HTTP (where responses to POST are not cacheable). In CoAP, cacheability depends on the response code, so that also a response to a POST request can be made cacheable through a 2.05 Content code.

When the input data is large or has a binary encoding, it is better to use POST requests with a proper Media Type for the input representation. A POST request is also more suitable, when the result is time-dependent and the latest result is expected (e.g., exchange rates).

6.2.4. Events as State

In event-centric paradigms such as pub/sub, events are usually represented by an incoming message that might even be identical for each occurrence. Since the messages are queued, the receiver is aware of each occurrence of the event and can react accordingly. For instance, in an event-centric system, ringing a door bell would result in a message being sent that represents the event that it was rung.

In resource-oriented paradigms such as REST, messages usually carry the current state of the remote resource, independent from the changes (i.e., events) that have lead to that state. In a naive yet natural design, a door bell could be modeled as a resource that can have the states unpressed and pressed. There are, however, a few issues with this approach. Polling is not an option, as it is highly unlikely to be able to observe the pressed state with any realistic polling interval. When using CoAP Observe with Confirmable notifications, the server will usually send two notifications for the event that the door bell was pressed: notification for changing from unpressed to pressed and another one for changing back to unpressed. If the time between the state changes is very short, the server might

drop the first notification, as Observe only guarantees only eventual consistency (see Section 1.3 of [RFC7641]).

The solution is to pick a state model that fits better to the application. In the case of the door bell - and many other event-driven resources - the solution could be a counter that counts how often the bell was pressed. The corresponding action is taken each time the client observes a change in the received representation.

In the case of a network outage, this could lead to a ringing sound 10 minutes after the bell was rung. Also including a timestamp of the last counter increment in the state can help to suppress ringing a sound when the event has become obsolete.

6.3. Server Push

Overall, a universal mechanism for server push, that is, change-of-state notifications and stand-alone event notifications, is still an open issue that is being discussed in the Thing-to-Thing Research Group. It is connected to the state-event duality problem and custody transfer, that is, the transfer of the responsibility that a message (e.g., event) is delivered successfully.

A proficient mechanism for change-of-state notifications is currently only available for CoAP: Observing resources [RFC7641]. It offers eventual consistency, which guarantees "that if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state". It intrinsically deals with the challenges of lossy networks, where notifications might be lost, and constrained networks, where there might not be enough bandwidth to propagate all changes.

For stand-alone event notifications, that is, where every single notification contains an identifiable event that must not be lost, observing resources is not a good fit. A better strategy is to model each event as a new resource, whose existence is notified through change-of-state notifications of an index resource (cf. Collection pattern). Large numbers of events will cause the notification to grow large, as it needs to contain a large number of Web links. Blockwise transfers [RFC7959] can help here. When the links are ordered by freshness of the events, the first block can already contain all links to new events. Then, observers do not need to retrieve the remaining blocks from the server, but only the representations of the new event resources.

An alternative pattern is to exploit the dual roles of IoT devices, in particular when using CoAP: they are usually client and server at

the same time. A client observer would subscribe to events by registering a callback URI at the origin server, e.g., using a POST request and receiving the location of a temporary subscription resource as handle. The origin server would then publish events by sending POST requests containing the event to the observer. The cancellation can be modeled through deleting the subscription resource. This pattern makes the origin server responsible for delivering the event notifications. This goes beyond retransmissions of messages; the origin server is usually supposed to queue all undelivered events and to retry until successful delivery or explicit cancellation. In HTTP, this pattern is known as REST Hooks.

In HTTP, there exist a number of workarounds to enable server push, e.g., long polling and streaming [RFC6202] or server-sent events [W3C.REC-html5-20141028]. Long polling as an extension that both server and client need to be aware of. In IoT systems, long polling can introduce a considerable overhead, as the request has to be repeated for each notification. Streaming and server-sent events (in fact an evolved version of streaming) are more efficient, as only one request is sent. However, there is only one response header and subsequent notifications can only have content. There are no means for individual status and metadata, and hence no means for proficient error handling (e.g., when the resource is deleted).

7. Security Considerations

This document does not define new functionality and therefore does not introduce new security concerns. We assume that system designers apply classic Web security on top of the basic RESTful guidance given in this document. Thus, security protocols and considerations from related specifications apply to RESTful IoT design. These include:

- o Transport Layer Security (TLS): [RFC5246] and [RFC6347]
- o Internet X.509 Public Key Infrastructure: [RFC5280]
- o HTTP security: Section 9 of [RFC7230], Section 9 of [RFC7231], etc.
- o CoAP security: Section 11 of [RFC7252]
- o URI security: Section 7 of [RFC3986]

IoT-specific security is mainly work in progress at the time of writing. First specifications include:

- o (D)TLS Profiles for the Internet of Things: [RFC7925]

Further IoT security considerations are available in [I-D.irtf-t2trg-iot-secons].

8. Acknowledgement

The authors would like to thank Mert Ocak, Heidi-Maria Back, Tero Kauppinen, Michael Koster, Robby Simpson, Ravi Subramaniam, Dave Thaler, Erik Wilde, and Niklas Widell for the reviews and feedback.

9. References

9.1. Normative References

- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
"Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-04 (work in progress), July 2017.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess,
"CoRE Resource Directory", draft-ietf-core-resource-directory-11 (work in progress), July 2017.
- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine , 2000.
- [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>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5590] Harrington, D. and J. Schoenwaelder, "Transport Subsystem for the Simple Network Management Protocol (SNMP)", STD 78, RFC 5590, DOI 10.17487/RFC5590, June 2009, <<https://www.rfc-editor.org/info/rfc5590>>.

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<https://www.rfc-editor.org/info/rfc5988>>.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, DOI 10.17487/RFC6202, April 2011, <<https://www.rfc-editor.org/info/rfc6202>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [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>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [W3C.REC-exi-20110310]
Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <<http://www.w3.org/TR/2011/REC-exi-20110310>>.

[W3C.REC-html5-20141028]

Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, T., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, October 2014, <<http://www.w3.org/TR/2014/REC-html5-20141028>>.

9.2. Informative References

[CollectionJSON]

Amundsen, M., "Collection+JSON - Document Format", February 2013, <<http://amundsen.com/media-types/collection/format/>>.

[I-D.ietf-core-coap-pubsub]

Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-02 (work in progress), July 2017.

[I-D.ietf-core-senml]

Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-10 (work in progress), July 2017.

[I-D.irtf-t2trg-iot-secons]

Garcia-Morchon, O., Kumar, S., and M. Sethi, "State-of-the-Art and Challenges for the Internet of Things Security", draft-irtf-t2trg-iot-secons-05 (work in progress), September 2017.

[IANA-CoAP-media]

"CoAP Content-Formats", n.d., <<http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>>.

[IANA-media-types]

"Media Types", n.d., <<http://www.iana.org/assignments/media-types/media-types.xhtml>>.

[RFC6763]

Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.

[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>>.

- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [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>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.

Appendix A. Future Work

- o Interface semantics: shared knowledge among system components (URI schemes, media types, relation types, well-known locations; see core-apps)
- o Unreliable (best effort) communication, robust communication in network with high packet loss, 3-way commit
- o Discuss directories, such as CoAP Resource Directory
- o More information on how to design resources; choosing what is modeled as a resource, etc.

Authors' Addresses

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
Zurich CH-8092
Switzerland

Email: kovatsch@inf.ethz.ch

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

Email: hartke@tzi.org

Thing-to-Thing Research Group
Internet-Draft
Intended status: Experimental
Expires: August 23, 2016

M. Koster
SmartThings
February 20, 2016

Model-Based Hypertext Language
draft-koster-t2trg-hypertext-language-00

Abstract

Interoperability for connected things is improving in a number of important areas, converging around Internet Protocol (IP) and internet design patterns. Hypermedia is becoming more common with web linking becoming part of a number of important standards.

However, there is still an interoperability gap in how application semantics are defined and used. Many organizations and industry alliances are defining vocabulary and taxonomy for application domains, independent of each other. These vocabularies are often bound to particular conceptual models, and are often semantically incompatible with each other. While it may be possible to adapt protocols and convert representations, it is difficult to develop a common application framework that works across ecosystems and domains.

This article proposes a method that can be reused across application domains and across ecosystems, to define a shared conceptual model and common vocabulary. A public resource is described which does for connected things what schema.org does for web commerce, to provide a community driven vocabulary and simple ontology that enables web scale interoperability between applications and connected things.

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 August 23, 2016.

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
2. Conceptual Models and Domain Specific Language	3
3. Reference Architecture	4
4. Terminology	5
5. Informative References	5
Author's Address	7

1. Introduction

Interoperability for connected things is improving with the adoption of Internet Protocol (IP) in a number of emerging standards and ecosystems. [OCF], [Thread] and [ZigBee], [BT-SIG], [LWM2M], [IPSO], [Weave], and WiFi connected devices.

Consolidation is being seen in the adoption of HTTP and CoAP [RFC7252] by these ecosystems. Many emerging standards support some form of web linking or resource directory [OCF], [LWM2M], [RFC6690], [I-D.ietf-core-resource-directory]). Use of [REST] is becoming common, particularly for web integration and more recently at the device API and local network level [OCF], [LWM2M]. There is ongoing work in providing hypermedia support [I-D.ietf-core-interfaces], [OCF], [W3C-WoT], [T2TRG]. Hypertext based systems can use link attributes to incorporate application semantics.

For application semantics, many organizations and alliances are creating new vocabularies bound to their own models. This can result in application incompatibility, for example an identifier like temperature may appear to be well known but is semantically incompatible across ecosystems and application domains.

2. Conceptual Models and Domain Specific Language

The proposed approach to semantic interoperability is to develop a common semantic structure and language upon which to develop domain specific vocabularies that can be used for semantic annotation of resources and services.

For the purpose of this discussion, a language is assumed to consist of a vocabulary which is bound to a conceptual model. Such a conceptual model constrains and defines the grammar and semantics of the language.

Conceptual models are layered, with core concepts forming a basis on which to build more sophisticated models. In the case of human language, we are used to a set of conceptual fragments like nouns, verbs, adjectives, and so on to express thoughts and feelings when we interact with one another.

For interacting with connected things, there is a conceptual model in common use that is proposed by [W3C-WoT]. Using this model a connected Thing is defined by and interacted with through its Events, Actions, and Properties. Events can be considered to be state changes within the thing we may be interested in. Actions can be invoked when we want the thing to change its state or the state of its environment, like turn on a light. Properties are like the current state of a thing or its static attributes.

Events, Actions, and Properties can form the basis of a simple, consistent conceptual framework upon which to build a general interaction language for things. We can ask a thing about its properties, we can observe events that change the state of the thing or indicate changes in its environment, and we can ask it to perform actions for us to change its state or its environment.

Semantic interoperability may be achieved through common definitions of cross domain meta models and domain specific models, and shared vocabulary to describe the events, actions, and properties of connected things. For example, in the connected lighting domain, we may model a luminary as a thing that has various optional capabilities, like brightness control, color control, on/off control, and measurement of energy consumed. A common set of events, actions, and properties can be defined for all connected lights that have particular capabilities. By doing so, we enable control of any light using a common set of application level controls and affordances.

3. Reference Architecture

Figure 1 shows an example system architecture for broad interoperability at web scale.

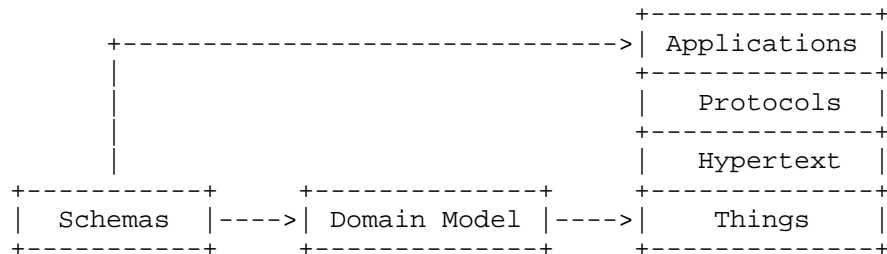


Figure 1: Reference Architecture

Schemas are publicly available domain specific meta-models, built through community consensus by domain experts and industry users. A working example of this today is [schemaorg], which is a public vocabulary and ontology for interoperable web commerce. Applications use schemas to develop machine comprehensions of the hypertext controls and affordances exposed by connected things.

Domain Models are built from schemas by manufacturers or data suppliers. Domain Models represent instances or specific types of products or information assets. A Domain Model would generally be used to construct a virtual thing or an interface to a thing.

Hypertext is exposed or referenced by things, and is used to bind vocabulary terms from the schema to instances of the Domain Model. Hypertext is made available in-band through the application protocol, for example CoAP or HTTP.

Protocols represent common network capability, for example [Thread], WiFi, Internet, and underlying protocols TCP/IP, UDP, and common application protocols CoAP, HTTP, as well as ecosystem protocol layers like [OCF] and [LWM2M].

Applications are the external logic and programs used to remotely control and orchestrate the activities of connected things. Domain specific vocabulary is exposed directly to applications through hypertext exposed through the application protocol.

Application level interoperability is achieved by applications understanding the semantics of the common underlying conceptual model and interpreting the vocabulary that describes domain specific

instances of that conceptual model. In the context of common application semantics, network and application protocols are relatively easy to adapt between ecosystems.

A reference implementation of the above architecture is described at [HypermediaDemonstrator].

4. Terminology

Ecosystem

Referring to a standard or platform supported by a corporate entity or an industry alliance or consortium.

Semantic Interoperability

The ability for data sources and applications to exchange state information in a meaningful way without prior detailed knowledge.

Hypertext, Hypermedia

Meta-data that identifies the location and attributes of resources.

Conceptual Model

A High level model that defines a set of semantic characteristics.

Schema:

A meta-model that defines how models are structured and annotated.

Domain Model

A model of a product or type that defines the expected configuration of a specific instance or type.

Application Domain

A group of applications that share common attributes and characteristics, for example connected automobiles.

5. Informative References

[BT-SIG] BT SIG, ., "Bluetooth SIG", 2016,
<<https://www.bluetooth.com/>>.

[HypermediaDemonstrator]

Koster, M., "Demonstration of a HATEOAS based system architecture for the Web of Things", 2016,
<<https://github.com/connectIOT/HypermediaDemo>>.

- [I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., and M. Koster, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-04 (work in progress), October 2015.
- [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-04 (work in progress), November 2015.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.
- [IPSO] IPSO Alliance, ., "IPSO Smart Object Guidelines", 2016, <<http://www.ipso-alliance.org/ipso-community/resources/smart-objects-interopability/>>.
- [LWM2M] OMA, ., "OMA LWM2M", 2016, <<http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0>>.
- [OCF] OCF, ., "Open Connectivity Foundation", 2016, <<http://openconnectivity.org/>>.
- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [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>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.

- [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>>.
- [schemaorg] org, Schema., "Schema.org", 2016, <<http://schema.org>>.
- [T2TRG] IRTF, ., "IRTF Thing to Thing Research Group", 2016, <<https://datatracker.ietf.org/rg/t2trg/charter/>>.
- [Thread] Thread Group, ., "Thread Group", 2016, <<http://www.threadgroup.org/>>.
- [W3C-WoT] WoT IG, ., "W3C Web of Things Interest Group", 2016, <<https://www.w3.org/WoT/IG/>>.
- [Weave] Nest, ., "Nest Weave Overview", 2016, <<https://developer.nest.com/documentation/weave/weave-overview/>>.
- [ZigBee] ZigBee Alliance, ., "ZigBee Cluster Library", 2016, <<http://www.zigbee.org/download/standards-zigbee-cluster-library/>>.

Author's Address

Michael Koster
SmarThings
1281 Lawrence Station Road
Sunnyvale 94089
USA

Phone: +1-707-502-5136
Email: michael.koster@smarththings.com

Thing-to-Thing Research Group
Internet-Draft
Intended status: Informational
Expires: August 25, 2016

K. Lynn
L. Dornin
Verizon Labs
February 22, 2016

Modeling RESTful APIs with JSON Hyper-Schema
draft-lynn-t2trg-model-rest-apis-00

Abstract

This document explores JSON Hyper-Schema as a method of modeling Internet of Things (IoT) systems that follow the principles of the Representational State Transfer (REST) architectural style.

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 August 25, 2016.

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
2. Overview of JSON Schema and Hyper-Schema	3
3. Examples	3
4. IANA Considerations	8
5. Security Considerations	8
6. References	8
Appendix A. Future Work	9
Authors' Addresses	9

1. Introduction

The central problem in an IoT domain such as home control might be characterized as "translating intention into configuration". The challenge is to translate a high level goal such as "turn off all the lights on the first floor", expressed in a natural language, into action. An agent trying to accomplish this goal might look up "light" in a semantic registry, discover instances of "light", select those instances that fall within a given area, etc., and ultimately transmit control packets to devices that match the given criteria.

A user defining such a goal should not be concerned with the underlying technology, i.e., details of the protocols or data links over which the control packets are sent. Similarly, service providers or developers wishing to provide home control solutions would generally prefer them to be technology agnostic. As with other translation tasks, this can more easily be accomplished by using one or more intermediate abstraction layers.

Recent research in semantic interoperability in heterogeneous environments has underscored the need for semantic alignment at various levels [Semantic-Interop]. This document is concerned with identifying a fairly primitive abstraction that sufficiently models target state and behavior without overly specifying the details of the underlying technology. We propose to explore REST APIs as a reasonable layer at which to abstract devices.

JSON Hyper-Schema [I-D.luff-json-hyper-schema] is a formalism for describing RESTful APIs. It supports a description of inbound and outbound data across the interface, together with link descriptions that identify the URIs, link-relations, and methods that apply to links. JSON Hyper-Schema therefore supports the principal of Hypertext as the Engine of Application State. It may also serve as input to documentation and code generation tools.

2. Overview of JSON Schema and Hyper-Schema

JSON Schema is a JSON based format for defining the structure of JSON data [RFC7159][I-D.zyp-json-schema]. JSON Hyper-Schema adds hyperlink- and hyper-media related keywords to JSON Schema. This section simply lists features of JSON Schema and Hyper-Schema used in the examples. For a detailed overview, see [Understanding-JSON-Schema]

2.1. JSON Schema

JSON Schema supports:

- JSON data types: object, array, number, string, boolean, null
- \$schema keyword: value identifies meta-schema and version
- Definitions and JSON references promote fragment reuse
- Schema composition keywords: oneOf, anyOf, allOf
- Patterns, regular expressions, more...

2.2. JSON Hyper-Schema

JSON Hyper-Schema adds Link Description Objects which include:

- href: URI template
- rel: link relation
- title: a title for the link
- targetSchema: JSON Schema describing the link target
- mediaType: media type describing the link target
- method: REST method that applies to this link
- encType: media type of the request
- schema: Schema describing the data sent with the request

3. Examples

The following examples may be validated at <http://json-schema-validator.herokuapp.com/>. The first example may be converted to markdown using the prmd tool at <https://github.com/interagent/prmd>.

3.1. Binary Switch Hyper-Schema

This JSON Hyper-Schema models a generic device on/off capability.

```
{
  "$schema": "http://json-schema.org/draft-04/hyper-schema#",
  "id": "http://example.com/schemata/switch-binary#",
  "description": "A simple API for a device that supports on/off.",
  "type": [
    "object"
  ],
  "definitions": {
    "uuid": {
      "description": "Unique identifier of the device",
      "example": "01234567-89ab-cdef-0123-456789abcdef",
      "format": "uuid",
      "readOnly": true,
      "type": [
        "string"
      ]
    },
    "identity": {
      "anyOf": [
        {
          "$ref": "#/definitions/uuid"
        }
      ]
    },
    "invalidated": {
      "description": "Time resource was invalidated",
      "example": "2015-01-01T12:00:00Z",
      "format": "date-time",
      "readOnly": true,
      "type": [
        "string"
      ]
    },
    "updated": {
      "description": "Time resource was last updated",
      "example": "2015-01-01T12:00:01Z",
      "format": "date-time",
      "readOnly": true,
      "type": [
        "string"
      ]
    },
    "SwBinary": {
      "title": "Binary Switch",
```

```
"description": "Used to control devices with On/Off capability.",
"stability": "prototype",
"type": [
  "object"
],
"definitions": {
  "SetValue": {
    "description": "0..99 (level) or 255 (on)",
    "example": 50,
    "type": "number",
    "multipleOf": 1,
    "oneOf": [
      {
        "minimum": 0,
        "maximum": 99
      },
      {
        "enum": [
          255
        ]
      }
    ]
  },
  "GetValue": {
    "description": "0 (off) or 255 (on)",
    "example": 255,
    "type": "number",
    "multipleOf": 1,
    "oneOf": [
      {
        "enum": [
          0,
          255
        ]
      }
    ]
  }
},
"properties": {
  "invalidated": {
    "$ref": "#/definitions/invalidated"
  },
  "updated": {
    "$ref": "#/definitions/updated"
  },
  "Value": {
    "$ref": "#/definitions/SwBinary/definitions/GetValue"
  }
}
```

```

    },
    "links": [
      {
        "title": "Set",
        "description": "Update a specific Binary Switch instance.",
        "href": "/id/{(%23%2Fdefinitions%2Fidentity)}/SwBinary/Set",
        "method": "POST",
        "rel": "update",
        "schema": {
          "type": [
            "object"
          ],
          "properties": {
            "Value": {
              "$ref": "#/definitions/SwBinary/definitions/SetValue"
            }
          },
          "required": [
            "Value"
          ],
          "strictProperties": true
        }
      },
      {
        "title": "Get",
        "description": "Read a specific Binary Switch instance.",
        "href": "/id/{(%23%2Fdefinitions%2Fidentity)}/SwBinary/Get",
        "method": "GET",
        "rel": "self",
        "targetSchema": {
          "$ref": "#/definitions/SwBinary"
        }
      }
    ]
  },
  "properties": {
    "SwBinary": {
      "$ref": "#/definitions/SwBinary"
    }
  },
  "additionalProperties": false,
  "links": [
    {
      "href": "https://",
      "rel": "self"
    }
  ],
  {

```

```
    "href": "/dev-schema",
    "method": "GET",
    "rel": "self",
    "targetSchema": {
      "additionalProperties": true
    }
  ]
}
```

3.2. JSON Link-Format Document

With the addition of the required "rel" property to each Link Description Object, the link-format example from section 2.4 of [I-D.ietf-core-links-json] becomes a valid JSON Hyper-Schema document.

```
[
  {
    "href": "/sensors",
    "ct": "40",
    "title": "SensorIndex",
    "rel": "self"
  },
  {
    "href": "/sensors/temp",
    "rt": "temperature-c",
    "if": "sensor",
    "rel": "self"
  },
  {
    "href": "/sensors/light",
    "rt": "light-lux",
    "if": "sensor",
    "rel": "self"
  },
  {
    "href": "http://www.example.com/sensors/t123",
    "anchor": "/sensors/temp",
    "rel": "describedby"
  },
  {
    "href": "/t",
    "anchor": "/sensors/temp",
    "rel": "alternate"
  }
]
```

4. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed upon publication as an RFC.

5. Security Considerations

This document doesn't define new functionality and therefore doesn't introduce new security concerns. However, security considerations from related specifications apply:

- o JSON security: section 12 of [RFC7159]

6. References

6.1. Normative References

- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [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, <<http://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, <<http://www.rfc-editor.org/info/rfc7252>>.

6.2. Informative References

- [I-D.luff-json-hyper-schema] Luff, G., Zyp, K., and G. Court, "JSON Hyper-Schema: Hypertext definitions for JSON Schema", draft-luff-json-hyper-schema-00 (work in progress), January 2013.
- [I-D.zyp-json-schema] Galiegue, F., Zyp, K., and G. Court, "JSON Schema: core definitions and terminology", draft-zyp-json-schema-04 (work in progress), January 2013.

- [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-04 (work in progress), November 2015.
- [REST]
Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [Understanding-JSON-Schema]
Droettboom, M., "Understanding JSON Schema, Release 1.0", Space Telescope Science Institute, February 2015, <<http://spacetelescope.github.io/understanding-json-schema/>>.
- [Semantic-Interop]
Konstantinos, K. and A. Katasonov, "Semantic Interoperability on the Web of Things: The Smart Gateway Framework", VTT Technical Research Center Tampere, Finland, DOI 10.1109/CISIS.2012.200, 2012, <https://www.researchgate.net/publication/231203029_Semantic_Interoperability_on_the_Web_of_Things_The_Smart_Gateway_Framework>.

Appendix A. Future Work

- o Provide more examples.
- o Discuss relationship to higher semantic layer(s).

Authors' Addresses

Kerry Lynn
Verizon Labs
50-60 Sylvan Rd
Waltham, MA 02451
USA

Phone: +1 781 296 9722
Email: kerlyn@ieee.org

Laird Dornin
Verizon Labs
50-60 Sylvan Rd
Waltham , MA 02451
USA

Phone: +1 781 466 2062
Email: laird.dornin@verizon.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 22, 2021

M. Sethi
Ericsson
B. Sarikaya
Denpel Informatique
D. Garcia-Carrillo
University of Oviedo
February 18, 2021

Secure IoT Bootstrapping: A Survey
draft-sarikaya-t2trg-sbootstrapping-11

Abstract

This draft provides an overview of the various terms that are used when discussing bootstrapping of IoT devices. We document terms that have been used within the IETF as well as other standards bodies. We investigate if the terms refer to the same phenomena or have subtle differences. We provide recommendations on the applicability of terms in different contexts. Finally, this document presents a survey of secure bootstrapping mechanisms available for smart objects that are part of an Internet of Things (IoT) network. The survey does not prescribe any one mechanism and rather presents IoT developers with different options to choose from, depending on their use-case, security requirements, and the user interface available on their IoT devices.

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 August 22, 2021.

Copyright Notice

Copyright (c) 2021 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 (<https://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	3
2. Terminology	4
3. Usage of bootstrapping terminology in standards	4
3.1. Device Provisioning Protocol (DPP)	4
3.2. Open Mobile Alliance (OMA) Lightweight M2M (LwM2M)	5
3.3. Open Connectivity Foundation (OCF)	6
3.4. Bluetooth	7
3.5. Fast IDentity Online (FIDO) alliance	7
3.6. Internet Engineering Task Force (IETF)	8
3.6.1. Enrollment over Secure Transport (EST)	8
3.6.2. Bootstrapping Remote Secure Key Infrastructures (BRSKI)	8
3.6.3. Secure Zero Touch Provisioning	8
3.6.4. Nimble out-of-band authentication for EAP (EAP-NOOB)	9
4. Comparison	9
5. Recommendations	9
6. Classification of available mechanisms	10
7. IoT Device Bootstrapping Methods	11
7.1. Managed Methods	11
7.1.1. Bootstrapping in LPWAN	13
7.2. Peer-to-Peer or Ad-hoc Methods	14
7.3. Leap-of-faith/Opportunistic Methods	15
7.4. Hybrid Methods	16
8. Security Considerations	16
9. IANA Considerations	17
10. Acknowledgements	17
11. Informative References	17
Authors' Addresses	23

1. Introduction

We informally define bootstrapping as "any process that takes place before a device can become operational". While bootstrapping is necessary for all computing devices, until recently, most of our devices typically had sufficient computing power and user interface (UI) for ensuring somewhat smooth operation. For example, a typical laptop device required the user to setup a username/password as well as enter network settings for Internet connectivity. Following these steps ensured that the laptop was fully operational.

The problem of bootstrapping is however exacerbated for Internet of Things (IoT) networks. The size of an IoT network varies from a couple of devices to tens of thousands, depending on the application. Smart objects/things/devices in IoT networks are produced by a variety of vendors and are typically heterogeneous in terms of the constraints on their power supply, communication capability, computation capacity, and user interfaces available. This problem of bootstrapping in IoT was identified by Sethi et al. [Sethi14] while developing a bootstrapping solution for smart displays. Although this document focuses on bootstrapping terminology and methods for IoT devices, we do not exclude bootstrapping related terminology used in other contexts.

Bootstrapping devices typically also involves providing them with some sort of network connectivity. Indeed, the functionality of a disconnected device is rather limited. Bootstrapping devices often assumes that some network has been setup a-priori. Setting up and maintaining a network itself is challenging. For example, users may need to configure the network name (called as Service Set Identifier (SSID) in Wi-Fi networks) and passphrase before new devices can be bootstrapped. Specifications such as the Wi-Fi Alliance Simple Configuration [simpleconn] help users setup networks. However, this document is only focused on terminology and processes associated with bootstrapping devices and excludes any discussion on setting up networks before devices can be bootstrapped.

In addition to our informal definition, it is helpful to look at other definitions of bootstrapping. The IoT@Work project defines bootstrapping in the context of IoT as "the process by which the state of a device, a subsystem, a network, or an application changes from not operational to operational" [iotwork]. Vermillard [vermillard], on the other hand, describes bootstrapping as the procedure by which an IoT device gets the URLs and secret keys for reaching the necessary servers. Vermillard notes that the same process is useful for re-keying, upgrading the security schemes, and for redirecting the IoT devices to new servers.

There are several terms that have often been used in the context of bootstrapping:

- o Bootstrapping
- o Provisioning
- o Onboarding
- o Enrollment
- o Commissioning
- o Initialization
- o Configuration
- o Registration

We attempt to find out whether all these terms refer to the same phenomena. We begin by looking at how these terms have been used in various standards and standardization bodies in Section 3. We then summarize our understanding in Section 4, and provide our recommendations on their usage in Section 5. Section 6 provides a taxonomy of bootstrapping methods and Section 7 categorizes methods according to the taxonomy.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174].

3. Usage of bootstrapping terminology in standards

To better understand bootstrapping related terminology, let us first look at the terms used by some existing specifications:

3.1. Device Provisioning Protocol (DPP)

The Wi-Fi Alliance Device provisioning protocol (DPP) [dpp] describes itself as a standardized protocol for providing user friendly Wi-Fi setup while maintaining or increasing the security. DPP relies on a configurator, e.g. a smartphone application, for setting up all other devices, called enrollees, in the network. DPP has the following three phases/sub-protocols:

- o **Bootstrapping:** The configurator obtains bootstrapping information from the enrollee using an out-of-band channel such as scanning a QR code or tapping NFC. The bootstrapping information includes the public-key of the device and metadata such as the radio channel on which the device is listening.
- o **Authentication:** In DPP, either the configurator or the enrollee can initiate the authentication protocol. The side initiating the authentication protocol is called as the initiator while the other side is called the responder. The authentication sub-protocol provides authentication of the responder to an initiator. It can optionally authenticate the initiator to the responder (only if the bootstrapping information was exchange out-of-band in both directions).
- o **Configuration:** Using the key established from the authentication protocol, the enrollee asks the configurator for network information such as the SSID and passphrase of the access point.

3.2. Open Mobile Alliance (OMA) Lightweight M2M (LwM2M)

The OMA LwM2M specification [oma] defines an architecture where a new device (LwM2M client) contacts a Bootstrap-server which is responsible for "provisioning" essential information such as credentials. After receiving this essential information, the LwM2M client device "registers" itself with one or more LwM2M Servers which will manage the device during its lifecycle. The current standard defines the following four bootstrapping modes:

- o **Factory Bootstrap:** An IoT device in this case is configured with all the necessary bootstrap information during manufacturing and prior to its deployment.
- o **Bootstrap from Smartcard:** An IoT device retrieves and processes all the necessary bootstrap data from a Smartcard.
- o **Client Initiated Bootstrap:** This mode provides a mechanism for an IoT client device to retrieve the bootstrap information from a Bootstrap Server. This requires the client device to have an account at the Bootstrap Server and credentials to obtain the necessary information securely.
- o **Server Initiated Bootstrap:** In this bootstrapping mode, the bootstrapping server configures all the bootstrap information on the IoT device without receiving a request from the client. This means that the bootstrap server needs to know if a client IoT Device is ready for bootstrapping before it can be configured.

For example, a network may inform the bootstrap server of a new connecting IoT client device.

3.3. Open Connectivity Foundation (OCF)

The Open Connectivity Foundation (OCF) [ocf] defines the process before a device is operational as onboarding. The first step of this onboarding process is "configuring" the ownership, i.e., establishing a legitimate user that owns the device. For this, the user is supposed to use an Onboarding tool (OBT) and an Owner Transfer Methods (OTM).

The OBT is described as a logical entity that may be implemented on a single or multiple entities such as a home gateway, a device management tool, etc. OCF lists several optional OTMs. At the end of the execution of an OTM, the onboarding tool must have "provisioned" an Owner Credential onto the device. The following owner transfer methods are specified:

- o Just works: Performs an un-authenticated Diffie-Hellman key exchange over Datagram Transport Layer Security (DTLS). The key exchange results in a symmetric session key which is later used for provisioning. Naturally, this mode is vulnerable to Man-in-The-Middle (MiTM) attackers.
- o Random PIN: The device generates a PIN code that is entered into the onboarding tool by the user. This pin code is used together with TLS-PSK ciphersuites for establishing a symmetric session key. OCF recommends PIN codes to have an entropy of 40 bits.
- o Manufacturer certificate: An onboarding tool authenticates the device by verifying a manufacturer installed certificate. Similarly, the device may authenticate the onboarding tool by verifying its signature.
- o Vendor specific: Vendors implement their own transfer method that accommodates any specific device constraints.

Once the onboarding tool and the new device have authenticated and established secure communication, the onboarding tool "provisions"/"configures" the device with Owner credentials. Owner credentials may consist of certificates, shared keys, or Kerberos tickets for example.

The OBT additionally configures/provisions information about the Access Management Service (AMS), the Credential Management Service (CMS), and the credentials for interacting with them. The AMS is

responsible for provisioning access control entries, while the CMS provisions security credentials necessary for device operation.

3.4. Bluetooth

Bluetooth mesh provisioning. Beacons for discovery. Public-key exchange followed by authentication. Finally provisioning of the network key and unicast address. To be expanded.

3.5. Fast IDentity Online (FIDO) alliance

The Fast IDentity Online Alliance (FIDO) is currently specifying an automatic onboarding protocol for IoT devices [fidospec]. The goal of this protocol is to provide a new IoT device with information for interacting securely with an online IoT platform. This protocol allows owners to choose the IoT platform for their devices at a late stage in the device lifecycle. The draft specification refers to this feature as "late binding".

The FIDO IoT protocol itself is composed of one Device Initialization (DI) protocol and 3 Transfer of Ownership (TO) protocols T00, T01, T02. Protocol messages are encoded in Concise Binary Object Representation (CBOR) [RFC8949] and can be transported over application layer protocols such as Constrained Application Protocol (CoAP) [RFC7252] or directly over TCP, Bluetooth etc. FIDO IoT however assumes that the device already has IP connectivity to a rendezvous server. Establishing this initial IP connectivity is explicitly stated as "out-of-scope" but the draft specification hints at the usage of Hypertext Transfer Protocol (HTTP) [RFC7230] proxies for IP networks and other forms of tunneling for non-IP networks.

The specification only provides a non-normative example of the DI protocol which must be executed in the factory during device manufacture. This protocol embeds initial ownership and manufacturing credentials into Restricted Operation Environment (ROE) on the device. The initial information embedded also includes a unique device identifier (called as GUID in the specification). After DI is executed, the manufacturer has an ownership voucher which is passed along the supply chain to the device owner.

When a device is unboxed and powered on by the new owner, the device discovers a network-local or an Internet-based rendezvous server. Protocols (T00, T01, and T02) between the device, the rendezvous server, and the new owner (as the owner onboarding service) ensure that the device and the new owner are able to authenticate each other. Thereafter, the new owner establishes cryptographic control of the device and provides it with credentials of the IoT platform which the device should use.

3.6. Internet Engineering Task Force (IETF)

In this section, we will look at some IETF standards and draft specifications related to IoT bootstrapping.

3.6.1. Enrollment over Secure Transport (EST)

Enrollment over Secure Transport (EST) [RFC7030] defines a profile of Certificate Management over CMS (CMC) [RFC5272]. EST relies on Hypertext Transfer Protocol (HTTP) and Transport Layer Security (TLS) for exchanging CMC messages and allows client devices to obtain client certificates and associated Certification Authority (CA) certificates. A companion specification for using EST over secure CoAP has also been standardized [I-D.ietf-ace-coap-est]. EST assumes that some initial information is already distributed so that EST client and servers can perform mutual authentication before continuing with protocol. [RFC7030] further defines "Bootstrap Distribution of CA Certificates" which allows minimally configured EST clients to obtain initial trust anchors. It relies on human users to verify information such as the CA certificate "fingerprint" received over the unauthenticated TLS connection setup. After successful completion of this bootstrapping step, clients can proceed to the enrollment step during which they obtain client certificates and associated CA certificates.

3.6.2. Bootstrapping Remote Secure Key Infrastructures (BRSKI)

The ANIMA working group is working on a bootstrapping solution for devices that relies on 802.1AR vendor certificates called Bootstrapping Remote Secure Key Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra]. In addition to vendor installed IEEE 802.1AR certificates, a vendor based service on the Internet is required. Before being authenticated, a new device only needs link-local connectivity, and does not require a routable address. When a vendor provides an Internet based service, devices can be forced to join only specific domains. The document highlights that the described solution is aimed in general at non-constrained (i.e. class 2+ defined in [RFC7228]) devices operating in a non-challenged network. It claims to scale to thousands of devices located in hostile environments, such as ISP provided CPE devices which are drop-shipped to the end user.

3.6.3. Secure Zero Touch Provisioning

[RFC8572] defines a bootstrapping strategy for enabling devices to securely obtain all the configuration information with no installer input, beyond the actual physical placement and connection of cables. Their goal is to enable a secure NETCONF [RFC6241] or RESTCONF

[RFC8040] connection to the deployment specific network management system (NMS). This bootstrapping method requires the devices to be configured with trust anchors in the form of X.509 certificates. [RFC8572] is similar to BRSKI based on [RFC8366].

3.6.4. Nimble out-of-band authentication for EAP (EAP-NOOB)

EAP-NOOB [I-D.ietf-emu-eap-noob] defines an EAP method where the authentication is based on a user-assisted out-of-band (OOB) channel between the server and peer. It is intended as a generic bootstrapping solution for IoT devices which have no pre-configured authentication credentials and which are not yet registered on the authentication server. This method claims to be more generic than most ad-hoc bootstrapping solutions in that it supports many types of OOB channels. The exact in-band messages and OOB message contents are specified and not the OOB channel details. EAP-NOOB also supports IoT devices with only output (e.g. display) or only input (e.g. camera). It makes combined use of both secrecy and integrity of the OOB channel for more robust security than the ad-hoc solutions.

4. Comparison

There are several stages before a device becomes fully operational. This typically involves establishing some initial trust after which credentials and other parameters are configured. For DPP, bootstrapping is the first step before authentication and provisioning of credentials can occur. For EST, bootstrapping happens as the first step when the client devices have no certificates available for starting enrollment. Provisioning/configuring are terms used for providing additional information to devices before they are fully operational. For example, credentials are provisioned onto the device. But before credential provisioning, a device is bootstrapped and authenticated. Some protocols may only deal with parts of the process. For example, TLS maybe used for authentication after bootstrapping. A separate device management protocol then may run over this TLS tunnel for provisioning operational information and credentials.

5. Recommendations

- o It is recommended that the IETF use the term "bootstrapping" for the initial (authentication) step that a device must perform. Bootstrapping will likely happen before the device has obtained full network connectivity.
- o It is recommended to use the term "provisioning"/"configuring" for the process of providing necessary information to a device to

become operational after initial authentication is complete. As is evident from above, provisioning and configuring may include bootstrapping and authentication as a sub protocol.

- o IETF specifications should aim to avoid mixing terminology or adding new terminology for better consistency.

6. Classification of available mechanisms

Given the large number of bootstrapping protocols and related specifications, it can be helpful to classify them. We categorize the available bootstrapping solutions into the following major classes:

- o **Managed methods:** These methods rely on pre-established trust relations and authentication credentials. They typically utilize centralized servers for authentication, although several such servers may join to form a distributed federation. Example methods include Extensible Authentication Protocol (EAP) [RFC3748], Generic Bootstrapping Architecture (GBA) [TS33220], Kerberos [RFC4120], Bootstrapping Remote Secure Key Infrastructures (BRSKI) and vendor certificates [vendorcert]. EAP Transport Layer Security EAP-TLS [I-D.ietf-emu-eap-tls13] for instance assumes that both the client and the server have certificates to authenticate each other. Based on this authentication, the server authorizes the client for network access. The Eduroam federation [RFC7593] uses a network of such servers to support roaming clients.
- o **Opportunistic and leap-of-faith methods:** In these methods, rather than verifying the initial authentication, the continuity of the initial identity or connection is verified. Some of these methods assume that the attacker is not present during the initial setup. Example methods include Secure Neighbor Discovery (SEND) [RFC3971] and Cryptographically Generated Addresses (CGA) [RFC3972], Wifi Protected Setup (WPS) push button [wps], and Secure Shell (SSH) [RFC4253].
- o **Peer-to-Peer (P2P) and Ad-hoc methods:** These bootstrapping methods do not rely on any pre-established credentials. Instead, the bootstrapping protocol results in credentials being established for subsequent secure communication. Such bootstrapping methods typically perform an unauthenticated Diffie-Hellman exchange [dh] and then use an out-of-band (OOB) communication channel to prevent a man-in-the-middle attack (MitM). Various secure device pairing protocols fall in this category. Based on how the OOB channel is used, the P2P methods can be further classified into two sub categories:

- * Key derivation: Contextual information received over the OOB channel is used for shared key derivation. For example, [proximate] relies on the common radio environment of the devices being paired to derive the shared secret which would then be used for secure communication.
- * Key confirmation: A Diffie-Hellman key exchange occurs over the insecure network and the established key is used to authenticate with the help of the OOB channel. For example, Bluetooth simple pairing [SimplePairing] use the OOB channel to ask the user to compare pins and approve the completed exchange.
- o Hybrid methods: Most deployed methods are hybrid and use components from both managed and ad-hoc methods. For instance, central management may be used for devices after they have been registered with the server using ad-hoc registration methods.

It is important to note here that categorization of different methods is not always easy or clear. For example, all the opportunistic and leap-of-faith methods become managed methods after the initial vulnerability window. The choice of bootstrapping method used for devices depends heavily on the business case. Questions that may govern the choice include: What third parties are available? Who wants to retain control or avoid work? In each category, there are many different methods of secure bootstrapping available. The choice of the method may also be governed by the type of device being bootstrapped.

7. IoT Device Bootstrapping Methods

In this section we look at additional bootstrapping protocols for IoT devices which are not covered in Section 3. Protocols already covered in Section 3 however are mentioned in their respective classes. This list is non-exhaustive.

7.1. Managed Methods

EAP-TLS is a widely used EAP method for network access authentication [I-D.ietf-emu-eap-tls13]. It requires certificate-based mutual authentication and a public key infrastructure. The ZigBee Alliance has specified an IPv6 stack for IEEE 802.15.4 [IEEE802.15.4] devices used in smart meters developed primarily for SEP 2.0 (Smart Energy Profile) application layer traffic [SEP2.0]. The ZigBee IP stack uses EAP-TLS for secure bootstrapping of devices.

EAP-PSK [RFC4764] is another EAP method that realizes mutual authentication and session key derivation using a Pre-Shared Key

(PSK). Given the light-weight nature of EAP-PSK, it can be suitable for resource-constrained devices. However, secure distribution of a large number of PSKs can be challenging.

CoAP-EAP [I-D.marin-ace-wg-coap-eap] defines a bootstrapping service for IoT. The authors propose transporting EAP over CoAP [RFC7252] for the constrained link, and communication with AAA infrastructures in the non-constrained link. While the draft discusses the use of EAP-PSK, the authors claim that they are specifying a new EAP lower layer and any EAP method which results in generation is suitable.

Protocol for Carrying Authentication for Network Access (PANA) [RFC5191] is a network layer protocol with which a node can authenticate itself to gain access to the network. PANA does not define a new authentication protocol and rather uses EAP over User Datagram Protocol (UDP) for authentication.

Colin O'Flynn [I-D.oflynn-core-bootstrapping] proposes the use of PANA for secure bootstrapping of resource constrained devices. He demonstrates how a 6LoWPAN Border Router (PANA Authentication Agent (PAA)) can authenticate the identity of a joining constrained device (PANA Client). Once the constrained device has been successfully authenticated, the border router can also provide network and security parameters to the joining device.

Hernandez-Ramos et al. [panaiot] also use EAP-TLS over PANA for secure bootstrapping of smart objects. They extend their bootstrapping scheme for configuring additional keys that are used for secure group communication.

Generic Bootstrapping Architecture (GBA) is another bootstrapping method that falls in centralized category. GBA is part of the 3GPP standard [TS33220] and is based on 3GPP Authentication and Key Agreement (3GPP AKA). GBA is an application independent mechanism to provide a client application (running on the User equipment (UE)) and any application server with a shared session secret. This shared session secret can subsequently be used to authenticate and protect the communication between the client application and the application server. GBA authentication is based on the permanent secret shared between the UE's Universal Integrated Circuit Card (UICC), for example SIM card, and the corresponding profile information stored within the cellular network operator's Home Subscriber System (HSS) database. [I-D.sethi-gba-constrained] describes a resource-constrained adaptation of GBA for IoT.

The four bootstrapping modes specified by the Open Mobile Alliance (OMA) Light-weight M2M (LwM2M) standard require some sort of pre-

provisioned credentials on the device. All the four modes are examples of managed bootstrapping methods.

The Kerberos protocol [RFC4120] is a network authentication protocol that allows several endpoints to communicate over an insecure network. Kerberos relies on a symmetric cryptography scheme and requires a trusted third party, that guarantees the identities of the various actors. It relies on the use of "tickets" for nodes to prove identity to one another in a secure manner. There has been research work on using Kerberos for IoT devices [kerberosiot].

It is also important to mention some of the work done on implicit certificates and identity-based cryptographic schemes [himmo], [implicit]. While these are interesting and novel schemes that can be a part of securely bootstrapping devices, at this point, it is hard to speculate on whether such schemes would see large-scale deployment in the future.

7.1.1. Bootstrapping in LPWAN

Low Power Wide Area Network (LPWAN) encompasses a wide variety of technologies whose link-layer characteristics are severely constrained in comparison to other typical IoT link-layer technologies such as Bluetooth or IEEE 802.15.4. While some LPWAN technologies rely on proprietary bootstrapping solutions which are not publicly accessible, others simply ignore the challenge of bootstrapping and key distribution. In this section, we discuss the bootstrapping methods used by LPWAN technologies covered in [RFC8376].

- o LoRaWAN [LoRaWAN] describes its own protocol to authenticate nodes before allowing them join a LoRaWAN network. This process is called as joining and it is based on pre-shared keys (called AppKeys in the standard). The joining procedure comprises only one exchange (join-request and join-accept) between the joining node and the network server. There are several adaptations to this joining procedure that allow network servers to delegate authentication and authorization to a backend AAA infrastructure [RFC2904].
- o Wi-SUN Alliance Field Area Network (FAN) uses IEEE 802.1X and EAP-TLS for network access authentication. It performs a 4-way handshake to establish a session keys after EAP-TLS authentication.
- o NB-IoT relies on the traditional 3GPP mutual authentication scheme based on a shared-secret in the Subscriber Identity Module (SIM) of the device and the mobile operator.

- o Sigfox security is based on unique device identifiers and cryptographic keys. As stated in [RFC8376], although the algorithms and keying details are not publicly available, there is sufficient information to indicate that bootstrapping in Sigfox is based on pre-established credentials between the device and the Sigfox network.

From the above, it is clear that all LPWAN technologies rely on pre-provisioned credentials for authentication between a new device and the network. Thus, all of them can be categorized as managed bootstrapping methods.

7.2. Peer-to-Peer or Ad-hoc Methods

While managed methods are viable for many IoT devices, they may not be suitable or desirable in all scenarios. All the managed methods assume that some credentials are provisioned into the device. These credentials may be in the device micro-controller or in a replaceable smart card such as a SIM card. The methods also sometimes assume that the manufacturer embeds these credentials during the device manufacture on the factory floor. However, in many cases the manufacturer may not have sufficient incentive to do this. In other scenarios, it may be hard to completely trust and rely on the device manufacturer to securely perform this task. Therefore, many times, P2P or Ad-hoc methods of bootstrapping are used. We discuss a few example next.

P2P or ad-hoc bootstrapping methods are used for establishing keys and credential information for secure communication without any pre-provisioned information. These bootstrapping mechanisms typically rely on an out-of-band (OOB) channel in order to prevent man-in-the-middle (MitM) attacks. P2P and ad-hoc methods have typically been used for securely pairing personal computing devices such as smart phones. [devicepairing] provides a survey of such secure device pairing methods. Many original pairing schemes required the user to enter the same key string or authentication code to both devices or to compare and approve codes displayed by the devices. While these methods can provide reasonable security, they require user interaction that is relatively unnatural and often considered a nuisance. Thus, there is ongoing research for more natural ways of pairing devices. To reduce the amount of user-interaction required in the pairing process, several proposals use contextual or location-dependent information, or natural user input such as sound or movement, for device pairing [proximate].

The local association created between two devices may later be used for connecting/introducing one of the devices to a centralized server. Such methods would however be classified as hybrids.

EAP-NOOB [I-D.ietf-emu-eap-noob] is an example of P2P and ad-hoc bootstrapping method that establishes a security association between an IoT device (node) and an online server (unlike pairing two devices for local connections over WiFi or Bluetooth).

Thread Group commissioning [threadcommissioning] introduces a two phased process i.e. Petitioning and Joining. Entities involved are leader, joiner, commissioner, joiner router and border router. Leader is the first device in Thread network that must be commissioned using out-of-band process and is used to inject correct user generated Commissioning Credentials (can be changed later) into Thread Network. Joiner is the node that intends to get authenticated and authorized on Thread Network. Commissioner is either within the Thread Network (Native) or connected with Thread Network via a WLAN (External).

Under some topologies, Joiner Router and Border Router facilitate the Joiner node to reach Native and External Commissioner, respectively. Petitioning begins before Joining process and is used to grant sole commissioning authority to a Commissioner. After an authorized Commissioner is designated, eligible thread devices can join network. Pair-wise key is shared between Commissioner and Joiner, network parameters (such as network name, security policy, etc.,) are sent out securely (using pair-wise key) by Joiner Router to Joiner for letting Joiner to join the Thread Network. Entities involved in Joining process depends on system topology i.e. location of Commissioner and Joiner. Thread networks only operate using IPv6. Thread devices can devise GUAs (Global Unicast Addresses) [RFC4291]. Provision also exist via Border Router, for Thread device to acquire individual global address by means of DHCPv6 or using SLAAC (Stateless Address Autoconfiguration) address derived with advertised network prefix.

7.3. Leap-of-faith/Opportunistic Methods

Bergmann et al. [simplekey] develop a secure bootstrapping mechanism that does not rely on pre-provisioned credentials using resurrecting-duckling imprinting scheme. Their bootstrapping protocol involves three distinct phases: discover (the duckling node searches for network nodes that can act as mother node), imprint (the mother node imprints a shared secret establishing a secure channel once a positive response is received for the imprinting request) and configure (additional configuration information such as network prefix and default gateway are configured). In this model for bootstrapping, a small initial vulnerability window is acceptable and can be mitigated using techniques such as a Faraday Cage (securing the communication physically) to protect the environment of the mother and duck nodes, though this may be inconvenient for the user.

7.4. Hybrid Methods

[RFC7250] defines how raw public keys can be used for mutual authentication of devices and servers. The extension specified in [RFC7250] simplifies `client_certificate_type` and `server_certificate_type` to carry only `SubjectPublicKeyInfo` structure with the raw public key instead of many other parameters found in typical X.509 version 3 certificates. Each side validates the keys received with pre-configured values stored. Using raw public keys for bootstrapping can be seen as a hybrid method. This is because it generally requires an out-of-band (OOB) step (P2P/Ad-hoc) where the raw public keys [RFC7250] are provided to the authenticating entities, after which the actual authentication occurs online (managed). CoAP already provides support for using raw public keys (see Section 9.1.3.2. of [RFC7252])

8. Security Considerations

This draft does not take any posture on the security properties of the different bootstrapping protocols discussed. Specific security considerations of bootstrapping protocols are present in the respective specifications.

Nonetheless, we briefly discuss some important security aspects which are not fully explored in various specifications.

Firstly, an IoT system may deal with authorization for resources and services separately from bootstrapping and authentication in terms of timing as well as protocols. As an example, two resource-constrained devices A and B may perform mutual authentication using credentials provided by an offline third-party X before device A obtains authorization for running a particular application on device B from an online third-party Y. In some cases, authentication and authorization maybe tightly coupled, e.g., successful authentication also means successful authorization.

Secondly, re-bootstrapping of IoT devices may be required since keys have limited lifetimes and devices may be lost or resold. Protocols and systems must have adequate provisions for revocation and re-bootstrapping. Re-bootstrapping must be as secure as the initial bootstrapping regardless of whether this re-bootstrapping is done manually or automatically over the network.

Lastly, some IoT networks use a common group key for multicast and broadcast traffic. As the number of devices in a network increase over time, a common group key may not be scalable and the same network may need to be split into separate groups with different keys. Bootstrapping and provisioning protocols may need appropriate

mechanisms for identifying and distributing keys to the current member devices of each group.

9. IANA Considerations

There are no IANA considerations for this document.

10. Acknowledgements

We would like to thank Tuomas Aura, Hannes Tschofenig, and Michael Richardson for providing extensive feedback as well as Rafa Marin-Lopez for his support.

11. Informative References

- [devicepairing] Mirzadeh, S., Cruickshank, H., and R. Tafazolli, "Secure Device Pairing: A Survey", IEEE Communications Surveys and Tutorials , pp. 17-40, 2014.
- [dh] Diffie, W. and M. Hellman, "New directions in cryptography", IEEE Transactions on Information Theory , vol. 22, no. 6, pp. 644-654, 1976.
- [dpp] Wi-Fi Alliance, "Wi-Fi Device Provisioning Protocol (DPP)", Wi-Fi Alliance , 2018, <https://www.wi-fi.org/download.php?file=/sites/default/files/private/Device_Provisioning_Protocol_Specification_v1.1_1.pdf>.
- [fidospec] Fast Identity Online Alliance, "FIDO IoT Spec", Fido Alliance , August 2020, <<https://fidoalliance.org/specs/internet-of-things/FIDO-IoT-spec.html>>.
- [himmo] Garcia-Morchon, O., Rietman, R., Sharma, S., Tolhuizen, L., and J. Torre-Arce, "DTLS-HIMMO: Efficiently Securing a Post-Quantum World with a Fully-Collusion Resistant KPS", Submitted to NIST Workshop on Cybersecurity in a Post-Quantum World , version 20141225:065757, December 2014, <<https://eprint.iacr.org/2014/1008>>.
- [I-D.ietf-ace-coap-est] Stok, P., Kampanakis, P., Richardson, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-est-18 (work in progress), January 2020.

- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M.,
and K. Watsen, "Bootstrapping Remote Secure Key
Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-
keyinfra-45 (work in progress), November 2020.
- [I-D.ietf-emu-eap-noob]
Aura, T., Sethi, M., and A. Peltonen, "Nimble out-of-band
authentication for EAP (EAP-NOOB)", draft-ietf-emu-eap-
noob-03 (work in progress), December 2020.
- [I-D.ietf-emu-eap-tls13]
Mattsson, J. and M. Sethi, "Using EAP-TLS with TLS 1.3",
draft-ietf-emu-eap-tls13-13 (work in progress), November
2020.
- [I-D.marin-ace-wg-coap-eap]
Marin-Lopez, R. and D. Garcia-Carrillo, "EAP-based
Authentication Service for CoAP", draft-marin-ace-wg-coap-
eap-07 (work in progress), January 2021.
- [I-D.oflynn-core-bootstrapping]
Sarikaya, B., Ohba, Y., Cao, Z., and R. Cragie, "Security
Bootstrapping of Resource-Constrained Devices", draft-
oflynn-core-bootstrapping-03 (work in progress), November
2010.
- [I-D.sethi-gba-constrained]
Sethi, M., Lehtovirta, V., and P. Salmela, "Using Generic
Bootstrapping Architecture with Constrained Devices",
draft-sethi-gba-constrained-01 (work in progress),
February 2014.
- [IEEE802.15.4]
"IEEE Std. 802.15.4-2015", April 2016,
<[http://standards.ieee.org/findstds/
standard/802.15.4-2015.html](http://standards.ieee.org/findstds/standard/802.15.4-2015.html)>.
- [implicit]
Porambage, P., Schmitt, C., Kumar, P., Gurtov, A., and M.
Ylianttila, "Pauthkey: A pervasive authentication protocol
and key establishment scheme for wireless sensor networks
in distributed iot applications", International Journal of
Distributed Sensor Networks , Hindawi Publishing
Corporation , 2014.
- [iotwork] European Commission FP7, "IoT@Work bootstrapping
architecture Deliverable D2.2", June 2011.

- [kerberosiot] Hardjono, T., "Kerberos for Internet-of-Things", February 2014, <https://kit.mit.edu/sites/default/files/documents/Kerberos_Internet_of%20Things.pdf>.
- [LoRaWAN] Sornin, N., Luis, M., Eirich, T., and T. Kramp, "LoRa Specification V1.0", January 2015, <<https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf>>.
- [ocf] Open Connectivity Foundation, "OCF Security Specification", Open Connectivity Foundation , June 2017, <https://openconnectivity.org/specs/OCF_Security_Specification_v1.0.0.pdf>.
- [oma] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification: Core", Open Mobile Alliance , November 2020, <www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Core-V1_2-20201110-A.pdf>.
- [panaiot] Hernandez-Ramos, J., Carrillo, D., Marin-Lopez, R., and A. Skarmeta, "Dynamic Security Credentials PANA-based Provisioning for IoT Smart Objects", 2nd World Forum on Internet of Things (WF-IoT) , IEEE , 2015.
- [proximate] Mathur, S., Miller, R., Varshavsky, A., Trappe, W., and N. Mandayam, "Proximate: proximity-based secure pairing using ambient wireless signals.", Proceedings of MobiSys International Conference , pp. 211-224, June 2011.
- [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>>.
- [RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, DOI 10.17487/RFC2904, August 2000, <<https://www.rfc-editor.org/info/rfc2904>>.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.

- [RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <<https://www.rfc-editor.org/info/rfc3971>>.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, DOI 10.17487/RFC3972, March 2005, <<https://www.rfc-editor.org/info/rfc3972>>.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <<https://www.rfc-editor.org/info/rfc4120>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<https://www.rfc-editor.org/info/rfc4253>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC4764] Bersani, F. and H. Tschofenig, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method", RFC 4764, DOI 10.17487/RFC4764, January 2007, <<https://www.rfc-editor.org/info/rfc4764>>.
- [RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <<https://www.rfc-editor.org/info/rfc5191>>.
- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.

- [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>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [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>>.
- [RFC7593] Wierenga, K., Winter, S., and T. Wolniewicz, "The eduroam Architecture for Network Roaming", RFC 7593, DOI 10.17487/RFC7593, September 2015, <<https://www.rfc-editor.org/info/rfc7593>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018, <<https://www.rfc-editor.org/info/rfc8376>>.
- [RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <<https://www.rfc-editor.org/info/rfc8572>>.

- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.
- [SEP2.0] ZigBee Alliance, "ZigBee IP Specification", March 2014, <<http://www.zigbee.org/non-menu-pages/zigbee-ip-download/>>.
- [Sethi14] Sethi, M., Oat, E., Di Francesco, M., and T. Aura, "Secure Bootstrapping of Cloud-Managed Ubiquitous Displays", Proceedings of ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2014), pp. 739-750, Seattle, USA, September 2014, <<http://dx.doi.org/10.1145/2632048.2632049>>.
- [simpleconn] Wi-Fi Alliance, "Wi-Fi Simple Configuration", Wi-Fi Alliance, 2019, <https://www.wi-fi.org/download.php?file=/sites/default/files/private/Wi-Fi_Simple_Configuration_Technical_Specification_v2.0.7.pdf>.
- [simplekey] Bergmann, O., Gerdes, S., and C. Bormann, "Simple Keys for Simple Smart Objects", Smart Object Security Workshop, IETF 83, March 2012.
- [SimplePairing] Bluetooth, SIG, "Simple pairing whitepaper", Technical report, 2007.
- [threadcommissioning] Thread Group, "Thread Commissioning", Thread Group, Inc., 2015.
- [TS33220] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA) (Release 14)", December 2016, <<http://www.3gpp.org/DynaReport/33220.htm>>.
- [vendorcert] IEEE std. 802.1ar-2009, "Standard for local and metropolitan area networks - secure device identity", December 2009.

[vermillard]

Vermillard, J., "Bootstrapping device security with
lightweight M2M", Appeared on blog at medium.com ,
February 2015.

[wps]

Wi-Fi Alliance, "Wi-fi protected setup", Wi-Fi Alliance ,
2007.

Authors' Addresses

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: mohit@piuha.net

Behcet Sarikaya
Denpel Informatique

Email: sarikaya@ieee.org

Dan Garcia-Carrillo
University of Oviedo
Oviedo 33207
Spain

Email: garciadan@uniovi.es