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>.

Table of Contents

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:

- a link context,
- a link relation type that identifies the semantics of the link (see Section 2.3.1),
- a link target, identified by a URI, and
- 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:

- 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 <img> element. When a Web browser processes this element, it automatically dereferences the "src" and renders the resulting image in place of the <img> element.

- 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