CoRE Application Descriptions
draft-hartke-core-apps-03

Abstract

The interfaces of RESTful, hypertext-driven applications consist of reusable components such as Internet media types and link relation types. This document defines a simple standard that application designers can use to describe the interface of their application in a structured way so that other parties can develop interoperable clients and servers or 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>.

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 15, 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
1. Introduction

Representational State Transfer (REST) [15] 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 greatly increase 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] has been created, an application-layer protocol that enables RESTful applications in Constrained-Node Networks [11], thus 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 style, a set of constraints needs to be maintained when designing new applications and their application programming interfaces (APIs). One of the fundamental principles is that "REST APIs must be hypertext-driven" [16]. This principle is often ignored by application designers who instead specify their APIs in terms of fixed URIs through some out-of-band mechanism, e.g., in an API documentation. 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. Violations of REST design principles like this result in APIs that may not be as scalable, extensible, and interoperable as promised by REST [19].

REST is intended for long-lived network-based applications that span multiple organizations [16]. Principled REST APIs require some design effort, as 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 desirable features in the Internet of Things.

To aid application designers in the design process, this document proposes CoRE Application Descriptions, a simple standard for describing the APIs of constrained, RESTful, hypertext-driven applications. CoRE Application Descriptions 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 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 application components in their own applications.
2. Application Descriptions

A CoRE Application Description is a named set of reusable components. It is comprised of:

- URI schemes that identify communication protocols,
- Internet media types that identify representation formats,
- link relation types that identify link semantics,
- form relation types that identify form semantics, and
- optionally, well-known locations.

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

2.1. URI Schemes

The foundation of a hypertext-driven REST API are the communication protocol(s) spoken between a client and a server. Although HTTP/1.1 [13] is by far the most common communication protocol for REST APIs, a REST API should typically not be dependent on any specific communication protocol.

The use of a particular protocol is guided by URI schemes [7]. URI schemes specify the syntax and semantics of URI references [1] found in links (Section 2.2.2) and forms (Section 2.2.3).

A URI scheme refers to a family of protocols, typically distinguished by a version number. For example, the "http" URI scheme refers to the three members of the HTTP family of protocols: HTTP/1.0 [10], HTTP/1.1 [13] and HTTP/2 [8]. The specific HTTP version is negotiated between the client and the server through version indicators in the protocol or the TLS application-layer protocol negotiation (ALPN) extension [17].

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

2.2. Internet Media Types

One of the most important aspect of hypertext-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 core of a CoRE application description should be one or more media types.
A media type identifies a versioned series of representation formats (Section 2.2.1); 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" set to "utf-8".

Media types can be further refined by structured type name suffixes (e.g., "+xml" appended to the base subtype name; see Section 4.2.8 of RFC 6838), or by subtype information embedded in the representations themselves (e.g., "xmlns" declarations in XML documents [12]). Structured type name suffixes should be preferred, because embedded subtype information cannot be negotiated (e.g., using the CoAP Accept option).

In CoAP, media types are combined with content coding information [14] to indicate the "content format" [23] of a representation. Content formats are assigned a numeric identifier that can be used instead of the (typically much longer) media type string in representation formats with space constraints.

A media type 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>.

2.2.1. 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 evolve over time. It is part of the responsibility of the designer of a new version of a format to try to insure both forward and backward compatibility: new documents should work reasonably (with some fallback) with old processors, and old documents should work reasonably with new processors [20].

Representation formats enable hypertext-driven applications when they support the expression of hypermedia controls: links (Section 2.2.2) and/or forms (Section 2.2.3).

It is beyond the scope of this document to specify how hypermedia controls can be expressed. The appropriate syntax, when available, is described by the representation format specification associated with each media type.

2.2.2. Links

A link is the primary means for a client to change application state, i.e., to navigate from one resource to another. A link is a typed connection between two resources [5], and is comprised of:

- a context (usually the current resource),
- a link relation type that identifies the semantics of the link (Section 2.3),
- a target resource URI, and
- optionally, attributes that further describe the link target.

A link can be viewed as a statement of the form "{context} has a {relation type} resource at {target URI}, which has {target attributes}" [5]. For example, the resource <http://example.com/> could have a "terms-of-service" resource at <http://example.com/tos>, which has the media type "text/html".

There are two special kinds of links:
o An embedding link is a link with the additional hint that it, when processed, should be substituted with a representation of the referenced resource. Thus, traversing an embedding link adds to the application 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 returned image in place of the `<img>` element.

o A templated link is a link where the client constructs the target resource 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.

Tempered 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).

2.2.3. Forms

A form is the primary means for a client to change resource state. It is comprised of:

o a context (usually the current resource),

o a form relation type that identifies the semantics of the form (Section 2.4),

o a target resource URI,

o a submission method (PUT, POST, PATCH, or DELETE), and

o a description of a representation that the service accepts as part of form submission. This description can be a set of form fields or simply a list of acceptable media types.

A form can be viewed as an instruction of the form "To {relation type} the {context}, make a {method} request to {target URI}". For example, to "update" the resource <http://example.com/config>, a client should make a PUT request to <http://example.com/config>.

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 change resource state.
2.3. Link Relation Types

A link relation type identifies the semantics of a link [5]. 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 current 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. Rather, they only describe how the current context is related to another resource.

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 relation type can use an extension relation type, which is a URI that uniquely identifies the relation type (similar to a URI used as an XML namespace names). For example, an application can use "http://example.com/foo" as link relation type without having to register it.

In order to minimize the overhead of link relation types in representation formats with space constraints, link relation types are assigned a small numeric identifier that can be used in place of the link relation type name (see also Section 7.2). For example, the link relation type "copyright" has the numeric identifier 12.

2.4. Form Relation Types

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

IANA maintains a list of registered link relation types at <TBD>.

Similar to link relation types, applications can use extension relation types when they don’t wish to register a relation type.

Form relation types are also assigned a numeric identifier that can be used in place of the form relation type name.

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 [18] defines a metadata document format for describing
hosts; 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 a path
prefix in "http" and "https" URIs for this purpose, "/.well-known/";
RFC 7252 [23] extends this concept to "coap" and "coaps" URIs.

IANA maintains a list of registered well-known URIs at

3. Template

Application name:

URI schemes:

Media types:

Link relation types:

Form relation types:

Well-known locations:

Interoperability considerations:

Security considerations:

Contact:

Author/Change controller:

4. Guidelines for Application Designers

This section provides guidelines and recommendations for designers of
constrained, RESTful, hypertext-driven applications.

4.1. URI Design

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 REST APIs to prescribe a fixed format for this
data. However, there are a number of problems with this practice [6], in particular if the application designer and the server owner are not the same entity.

Therefore, in hypertext-driven applications URIs are not included in the application interface. A CoRE Application Description must not mandate any particular form of URI substructure that is not explicitly allowed by RFC 3986 [1].

RFC 7320 [6] describes the problematic practice of fixed URI structures in detail and provides some acceptable alternatives. Section 5.2 gives some recommendations on URI design for application implementers.

5. Guidelines for Application Implementers

This section provides guidelines and recommendations for implementers of application descriptions, on both the client and the server side.

5.1. Resource Navigation

Links (Section 2.2.2) and forms (Section 2.2.3) afford a potential next or future request from the client to a server. To make the request, the client needs to select a link or a form from a set of candidates. The process is as follows:

1. The first step for a client is to decide what to do next, i.e., which link to follow or which form to submit, based on the link relation types (Section 2.3) and form relation types (Section 2.4) it understands.

2. The client finds the link(s) or form(s) with the given relation type. This may yield one or more candidates from which the client must select the most appropriate one. The set of candidates may be empty if the transition is not allowed, for example, when the client is unauthorized.

3. The client selects one of the candidates based on the metadata associated with the hypermedia control. This includes the media type of the target resource representation, the URI scheme, the request method, and other attributes that describe the link target.

4. The client resolves the target resource URI reference to its absolute form in order to obtain the "request URI". The process for reference resolution is specified in Section 5 of RFC 3986 [1].
5. The client constructs the request. The request URI is the target resource URI resolved in the previous step. The request method is either GET for links or indicated by the hypermedia control for forms. The client should set request metadata according to the link/form attributes (e.g., set the CoAP Accept option when the media type of the target resource is specified). In case of a form, the client also constructs a request payload according to the instructions specified in the form.

6. Finally, the client sends the request and retrieves the response. The application thereby transitions to a new state and the client can decide again what to do next.

5.2. URI Design

Even though URIs are not part of the application interface (see Section 4.1), the design of the URI structure on a server is an essential part of implementing a RESTful application. The server implementer is responsible for binding the nouns identified by the application designer to URIs.

A good RESTful URI is:

- Short. Short URIs are easier to remember and cause less overhead in requests and representations.

- Meaningful. A URI should describe the resource in a way that is meaningful and useful to humans.

- Consistent. URIs should follow a consistent pattern to make it easy to reason about the application.

- Bookmarkable. Cool URIs don’t change [9]. However, application resource structures change. That should naturally cause URIs to change so they better reflect reality. So implementations should not depend on unchanging URIs.

- 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.

- 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).
6. Security Considerations


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.

7. IANA Considerations

[Note to RFC Editor: Please replace XXXX in this section with the RFC number of this specification.]

7.1. Content-Format Registry

RFC 6838 [2] establishes a IANA registry for media types. Many of these media types are also useful in constrained environments as CoAP content formats. RFC 7252 [23] establishes a IANA registry for these content formats. This specification tasks IANA with the allocation of a content format for any existing or new media type registration that does not define any parameters (required or optional). The content formats shall be allocated in the range 1000-9999.

7.2. Link Relation Type Registry

RFC 5988 [5] establishes a IANA registry for link relation types. This specification extends the registration template with a "Relation ID": a numeric identifier that can be used instead of the "Relation Name" to identify a link relation type. IANA is tasked with the assignment of an ID to any existing or new link relation type. The IDs shall be assigned in the range 1-9999.

7.3. Form Relation Type Registry

This specification establishes a IANA registry for form relation types.

7.3.1. Registering New Form Relation Types

Form relation types are registered in the same way as link relation types [5], i.e., they are registered on the advice of a Designated Expert with a Specification Required.

The requirements for registered relation types are adopted from Section 4.1 of RFC 5988 [5].
The registration template is:

- Relation Name:
- Relation ID:
- Description:
- Reference:
- Notes: [optional]

The IDs shall be assigned in the range 1-9999.

7.3.2. Initial Registry Contents

The Form Relation Type registry’s initial contents are:

- Relation Name: create-item
  Relation ID: 1
  Description: Refers to a resource that can be used to create a resource in a collection of resources.
  Reference: [RFCXXXX]

- Relation Name: delete
  Relation ID: 2
  Description: Refers to a resource that can be used to delete a resource in a collection of resources.
  Reference: [RFCXXXX]

- Relation Name: update
  Relation ID: 3
  Description: Refers to a resource that can be used to update the state of the form’s context.
  Reference: [RFCXXXX]

8. References

8.1. Normative References

Informative References


[23] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252,
DOI 10.17487/RFC7252, June 2014,

Acknowledgements

Thanks to Jan Algermissen, Mike Amundsen, Olaf Bergmann, Carsten Bormann, Stefanie Gerdes, Mike Kelly, Ari Keranen, Michael Koster, Matthias Kovatsch, Julian Reschke, Teemu Savolainen, Bilhanan Silverajan, and Erik Wilde for helpful comments and discussions that have shaped the document.

Some of the text in this document has been borrowed from [5], [6], [16], [19], and [20]. All errors are my own.

This work was funded in part by Nokia.

Author’s Address

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

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