Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 20, 2014

E. Wilde
EMC Corporation
September 16, 2013

HTTP Link Descriptions
draft-wilde-link-desc-00

Abstract

Interactions with many resources on the Web are driven by links, and these links often define certain expectations about the interactions (such as HTTP methods being used, media types being sent in the request, or URI parameters being used in a certain way). While these expectations are essential to define the possible framework for interactions, it may be useful to further narrow them down by providing link descriptions, which can help clients to gain more runtime knowledge about the resource they are about to interact with. This memo defines Link Descriptions, a model and associated media type that can be used to describe links by supporting descriptive markup for representing interaction information with links. Link Descriptions can be used by media types (by inclusion or by reference) that seek to make Link Descriptions runtime-capable, without having to create their own representation.

Note to Readers

Please discuss this draft on the apps-discuss mailing list [1].

Online access to all versions and files is available on github [2].

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

Interactions with resources found on the Web often are driven by following links (targeted at URIs [RFC3986]), which can be either fixed links (described in Section 1.1), or can be templated links (using URI Templates [RFC6570]) containing variables (described in Section 1.2). In both cases, the context of the link in most cases provides information that can be essential or helpful when it comes to following a link, which essentially means interacting with the link target: For fixed links, the context may provide (in most cases implicitly, through the use of typed links) allowed interaction methods (such as HTTP verbs) or expectations around the expected media type(s) in requests; for templated links, the context additionally may provide information about how to instantiate the variables provided in the URI Template. This memo defines a schema and a media type that can be used to (partially) represent this information, so that it becomes possible to represent a change in interaction affordances at runtime.

Possible use cases for both scenarios (fixed and templated links) are as follows:

Fixed Links: AtomPub [RFC5023] defines an "edit" link relation, that informs clients that a link can be followed to read, update, or delete a resource. This means that a client encountering such a link would conclude that it can try to read, update, or delete the target resource. However, if the resource is not deletable, then an "edit" link could be annotated to indicate that the linked resource cannot be deleted. A client could ignore the annotation and still attempt to delete the resource, but the request would be likely to fail (unless the state of the resource changed in the meantime). This kind of information can be very useful for UIs, where it can be used to drive usability features such as disabling certain UI elements.

Templated Links: URI Templates [RFC6570] define a framework for how to represent and instantiate (with concrete variable values) templated URIs, but they don't describe how variables themselves are described, or can be constrained. If a collection resource for example supports paged access to the set of collection members, then it might be useful for a client to know the number of available pages. With this additional knowledge, it is possible to build applications and UIs that specifically take this knowledge into account to drive further interactions with the resource. For a paged collection, it may be a UI that provides direct links to all available pages (if that number is reasonably small). Again, if the collection changes in size between the link being generated, and the link interaction taking place, the

information in the link description has become outdated. But this just means that either a client may request a page that doesn't exist anymore, or will not expect a page to exist that now exists. Both of these conditions can be handled well at the time when the client starts interacting with the linked resource.

As described in both cases, it is possible for the link description to become outdated, leading to cases where the assumptions made by the client (based on the link description) and the link target itself do not match anymore. For this reason, ideally a resource should provide a (link) description for itself, allowing a client to update its expectations. However, since the service generating the link and the service providing the link target are loosely coupled, link descriptions can be used in links, in descriptions where services expose more runtime information about resources by providing link descriptions for themselves, or in both places.

The following example shows hows both of these mechanisms can be used in one representation, which is based on Atom [RFC4287] and AtomPub [RFC5023]. It also shows the two cases just described, with the first link description being one to "self", while the second link description is about a different resource.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:ld="urn:ietf:rfc:XXXX" >
    <title>Example Feed</title>
    <link rel="self" href="http://example.org/?page=3" ld:hreft="http://
example.org/{?page}">
        <ld:var name="page" concept="http://example.com/feedpaging/page"
default="1">
            <ld:restriction base="positiveInteger">
                <ld:minInclusive value="1"/>
                <ld:maxInclusive value="42"/>
            </ld:restriction>
        </ld:var>
    </link>
    <link rel="next" href="http://example.org/?page=4"/>
    <link rel="previous" href="http://example.org/?page=2"/>
    <updated>2003-12-13T18:30:02Z</updated>
    <author>
        <name>John Doe</name>
    </author>
    <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
    <entry>
        <title>Atom-Powered Robots Run Amok</title>
        <link href="http://example.org/2003/12/13/atom03"/>
        <link rel="edit" href="http://example.org/item42">
            <ld:hint name="allow" value=' [ "PUT" ] '/>
            <ld:hint name="formats" value=' { "image/png" : {} , "image/jpeg" :
{} } '/>
        </link>
        <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
        <updated>2003-12-13T18:30:02Z</updated>
        <summary>Some text.</summary>
    </entry>
</feed>
```

The link to the feed itself is augmented with a URI Template described in Section 1.2, which allows a client to understand that individual feed pages can be requested (assuming the consumer understands the "concept" identifiers for the described variables). The link to the entry is an augmented typed Web Link described in Section 1.1, which allows a consumer to understand that even though "edit" links typically can be followed via GET, PUT, and DELETE, this particular link should only be followed using a PUT request.

It is worth noting that link descriptions of course can become outdated between the time such a link decription has been received by a client, and the time a client actually sends a request when following such a link (this is the case both for "self" links and links to other resources). This means that clients should never depend on link description being correct, because for example the "edit" link description shown above might start allowing DELETE requests again at any point in time.

1.1. Web Links

One of the defining principles of many services provided on the Web is that they expose linked resources, so that clients can follow the links in order to accomplish application goals. "Web Linking" [RFC5988] establishes a framework of typed links, allowing resources to expose typed links, which then can be followed by clients. While this framework allows clients to select links based on their types, it does not provide any support for additional runtime information about possible interactions with such a link. As outlined in the AtomPub example above, a link typed as "edit" (as defined and registered by AtomPub) by definition can be followed by using HTTP GET, PUT, or DELETE, and the typed link by itself cannot provide the additional information that some resource may allow updates, but disallows deletion.

"Link Hints" [I-D.nottingham-link-hint] provide a framework of runtime hints that can be used to indicate information that might be made available by the link target resource itself, but ahead of time. For example, a link hint would be able to indicate on an "edit" link that the resource only allows PUT requests, which is something that could also be discovered by sending an HTTP request and getting an HTTP Allow header in the response. However, link hints can save overhead by avoiding round trips, and they also allow to minimize the chances of sending requests that will not succeed.

While link hints can help to avoid overhead and drive client behavior, they are strictly optional. There should be no functional difference of what a client can achieve by using or ignoring link hints; they simply expose information that otherwise would be more costly to acquire.

Since it is potentially expensive to provide link hints in representations (because they may involve interpreting access control data), it is perfectly possible that services provide link hints only on some requests. For example, it would be possible to design a service that served http://example.com/collection as a collection of items with embedded "edit" links, whereas http://example.com/collection?hints=true would result in a representation that would contain additional link hints for each individual "edit" link. This kind of design is outside of the scope of this memo, but it's helpful to illustrate the fact that link hints are nothing but optimizations of at which point during interactions certain information is provided.

Currently, there is an overlap in what "Link Hints" [I-D.nottingham-link-hint] define, and what is proposed in this memo. Removing this overlap is captured in the "Open Issues" Section 9 and

should be addressed during the development of both drafts.

1.2. URI Templates

While following links is the basic principle of interacting with resources on the web, in many cases, interactions with resources require clients to provide information in addition to just using a fixed URI in a request. In these cases, information can be provided in any way supported by the interaction protocol, and in case of HTTP, this often means that information is either embedded in the URI, and/or in the body of the request. For the first case, "URI Template" [RFC6570] provides a standard that allows servers and clients to exchange information about the URIs that a service accepts. The standard specifies "a compact sequence of characters for describing a range of Uniform Resource Identifiers through variable expansion." It allows servers to publish their expectation how a URI should be created by substituting variables with values. Consider the following URI Template:
http://www.example.com/collection{?pagesize,page}

This URI Template allows clients to expand it with two variables values, to end up with a concrete URI such as the following:
http://www.example.com/collection?pagesize=10&page=42

URI Templates cover the aspect of starting with a template with variables in it, assigning values to these variables, and then expanding the template into a URI that can be used for sending a request. URI Templates make no assumptions or statements about the value range of the variables, except for those aspects which are required to cover the process of expanding the template. In particular, for the example given above, there is no indication that the values are supposed to be positive integers (the simple data type), nor is there any indication that the service may apply certain limits such as a maximum page size (which may change depending on which paged resource is being accessed). As a side note, even if this basic type information was known, URI template expansion could still result in URIs that would not yield successful requests, such as when asking for a page that is beyond the number of pages that a collection has (in a given page size).

The goal of Link Descriptions as defined in this memo is to allow servers to expose a description that provides support both at development time (when a developer looks at a media type that uses URI Templates) and at runtime (when a client wants to use a URI template as part of its application flow). Link Descriptions are intended to provide additional information that is not communicated by publishing URI Templates alone. The additional information is both targeted at machines and at humans. On the human-oriented Web,

a Template Description can be seen as the equivalent of a help or documentation page that is linked to from a form, where users can learn more about the values they are supposed to submit within the form.

As a concrete example, a link to a collection like the one above may be exposed in a link description as follows:

```
<link rel="items" href="http://www.example.com/collection" hreft="http://
www.example.com/collection{?pagesize,page}">
  <var name="pagesize" concept="http://example.com/feedpaging/pagesize"/>
  <var name="page" concept="http://example.com/feedpaging/page"/>
</link>
```

This link description allows a URI Template's variables to be described in terms of URI-identified concepts. By using such a model, it is possible to use global names for URI Template parameters, by binding them to the local variable names. The concept URIs are pure identifiers for the purpose of link descriptions; i.e. they should not be considered dereferenceable, and the assumption is that consumers of link descriptions will only use them to match discovered concepts against known concepts. This design does not prohibit to make concept URIs dereferenceable, but this is outside of the scope of link descriptions.

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 RFC 2119 [RFC2119].

3. Description Concepts

The general idea of link descriptions is that they allow to annotate links (URIs or URI Templates) with context that clients can use when they choose to follow those links. In the XML syntax, descriptions are deliberately designed to echo the design of Atom's popular <link> element, which serves as a blueprint for links in many media types. The main idea of link descriptions is to provide a framework which provides services that want to serve this kind of description with a starting point. If these services want, they can reuse the representations for this framework, in whole or in part.

3.1. Link Hints

As mentioned already, "Link Hints" [I-D.nottingham-link-hint] as currently defined overlap with the concepts proposed in this memo. However, this memo goes further that link hints by not just providing

hints for URIs, but for URI Templates as well. Based on the current link hint model, a link hint is a name/value pair, where the name is either a registered link hint, or a URI. The allowed value space depends on the link hint, and in the current model, structured values must be encoded in JSON. A hint may also contain application specific information or documentation, in an model of application information and documentation that has been inspired by XML Schema Part 1 [XSD-1].

A link may have any number of link hints, but only one link with a given name.

3.2. Describing Variables

When a link uses a URI Template, then this template will very likely contain variables. Variables can be described in a variety of ways when using Link Descriptions. For each variable contained in the URI Template, it is possible to use the following description methods:

Concept: It is possible to associate a variable with a concept, so that media types and applications can make an association between the concepts they are defining/exposing, and how they are exposed in URI Templates. Concepts can be identified by using a URI as an identifier. This specification defines no interactions with this URI identifier and makes no assumption about possible representations, should this URI be dereferenceable and yield some representation.

Datatype: Variables can be described in terms of using certain datatypes. The datatype vocabulary is that of XML Schema Part 2 [XSD-2], plus all of the applicable facets of those datatypes. This allows Link Descriptions to constrain the set of allowed values. (This model does not cover any "co-constraints", i.e. dependencies across variables, or between variables and an external context.)

Documentation: Documentation constructs can be associated with variables, which allows Link Descriptions to attach human-readable information to individual variables. The documentation constructs use the documentation design of XML Schema Part 1 [XSD-1]. XML Schema's documentation model has the ability to support multi-lingual human-oriented documentation.

Application Information: Application information constructs can be associated with variables, which allows Link Descriptions to attach machine-readable information to individual variables. The application information constructs use the application information design of XML Schema Part 1 [XSD-1].

For the purpose of this specification, the term "description" should be interpreted loosely. Some aspects of descriptions can be formal, such as the datatypes of variables. Thus, such a description can be used to drive general-purpose logic that knows no additional framework other than this specification. However, for most other description aspects (concepts, documentation, and application information), this specification does not prescribe a description framework; it simply provides a structure how to deliver these descriptions.

The descriptions of "concepts" and "datatypes" are not mutually exclusive. On the contrary, a concept often will have associated constraints that essentially define a datatype. The difference is that while concepts are only described by identifier (allowing a consumer to identify a concept that they know about), datatypes are described with a built-in vocabulary (XSD datatypes and their facets), allowing clients to interpret the datatype description. As a consequence, when a consumer encounters an unknown concept, it adds little to the knowledge of the consumer, whereas a datatype description can always be interpreted and thus allows the consumer to learn about constraints of the associated variable in a declarative way. It may therefore make sense to combine concepts and datatypes, if the goal is to be as self-describing as possible.

4. Link Descriptions

Link Descriptions are based on a URI Template, and add descriptive elements that allow publishers of URI Templates to describe the URI Template as a whole, and to add individual descriptions of all variables in the template. The idea of Link Descriptions is that they are made available at design time and/or at runtime, so that clients encountering URI Templates as part of HTTP services can find more information about the template itself.

Ideally, every URI template exposed in an HTTP service should be accompanied by a link to a Link Description. In those XML-based HTTP services where URI Templates are exposed in XML attributes named "hreft", the suggestion is to add a link to the corresponding Link Description in an "hrefd" XML attribute.

4.1. General Concepts

As mentioned in Section 1.2, most of the descriptions in this spec do not prescribe a specific description framework. While variables (Section 4.3) can be described with a built-in vocabulary of datatypes, most other descriptions are either for human consumption, or do rely on some external description framework. To attach these

descriptions to both the template as a whole, and individual variables, this specification reuses the "appinfo" and "documentation" elements from XML Schema Part 1 [XSD-1]. These elements carry a "source" attribute, which is used (quoting from [XSD-1]) "to supplement the local information." For example, when a description of a variable is done formally using a specific description framework, this would best translate to use appinfo elements, and to add an identifier to them which would identify the description framework in question. As a result, any client knowing this particular description framework would be able to interpret the variable description in the Link Description.

4.2. Link Description Structure

An interaction is described by including the URI Template itself, and optionally adding documentation and/or appinfo elements to add human- or machine-readable descriptions.

4.3. Variable Description Structure

A variable is described by specifying the variable name. Variables can refer to a "concept" associated with a variable, which can by identified by URI. This specification makes no provision how such a concept is defined and/or described/documented, but it allows consumers of a Link Description to match their understanding of certain concepts to those identifiers, which then establishes a binding between the concept, and the variable it has been bound to.

A variable can have a default value, in which case the assumption is that excluding this variable from a request has the same effect as including it with the default value. Since Link Descriptions are runtime concepts, however, there is no guarantee that a service might not use a different value between the time when the Link Description was retrieved, and the time when a request based on it is being sent.

Variable descriptions can optionally add documentation and/or appinfo elements to add human- or machine-readable descriptions.

5. Examples

...

5.1. Editable Entry

...

All the example use "documentation" elements which are entirely

   optional, but can help to improve the usefulness of link descriptions
   for developers.

```
<link xmlns="urn:ietf:rfc:XXXX" href="http://example.org/item42">
   <hint name="allow" value=' [ "PUT" ] '>
      <documentation xml:lang="en">For this particular resource, only PUT is
supported. Specifically, this resource does not accept DELETE requests.</
documentation>
   </hint>
   <hint name="formats" value=' { "image/png" : {} , "image/jpeg" : {} } '>
      <documentation xml:lang="en">Updates are accepted as PNG or JPEG
representations.</documentation>
   </hint>
   <documentation xml:lang="en">Template for accessing an AtomPub media
resource http://www.example.com/feed/item42, with the "edit-media" link by
default allowing PUT/DELETE as per RFC 5023.</documentation>
</link>
```

5.2. Pageable Collection

   ...

```
<link xmlns="urn:ietf:rfc:XXXX" hreft="http://example.org/{?pagesize,page}">
   <var name="pagesize" concept="http://example.com/feedpaging/pagesize"
default="10">
      <restriction base="positiveInteger">
         <minInclusive value="1"/>
         <maxInclusive value="100"/>
      </restriction>
      <documentation xml:lang="en">Number of returned items per page.</
documentation>
   </var>
   <var name="page" concept="http://example.com/feedpaging/page" default="1">
      <restriction base="positiveInteger"/>
      <documentation xml:lang="en">Page number of the returned page (based on
the requested pagesize or a service-defined default).</documentation>
   </var>
   <documentation xml:lang="en">Template for accessing a paged feed of entries
at http://www.example.com/feed, with client controls for the page size, and the
returned page.</documentation>
</link>
```

6. IANA Considerations

6.1. Media Type

   The Internet media type [RFC6838] for a Link Description document is
   application/ldesc+xml (using the "+xml" suffix as defined and
   registered by RFC 6839 [RFC6839]).

Type name: application

Subtype name: ldesc+xml

Required parameters: none

Optional parameters: profile

The "profile" link relation [RFC6906] allows "resource representations to indicate that they are following one or more profiles. A profile is defined not to alter the semantics of the resource representation itself, but to allow clients to learn about additional semantics (constraints, conventions, extensions) that are associated with the resource representation, in addition to those defined by the media type and possibly other mechanisms." If the application/ldesc+xml media type is use with a profile parameter, this refers to a profile as defined by [RFC6906], making it easier for extensions of the link description media type to identify themselves.

Encoding considerations: Same as encoding considerations of application/xml as specified in [RFC3023].

Security considerations: This media type has all of the security considerations described in [RFC3023], plus those listed in Section 8.

Interoperability considerations: N/A

Published specification: RFC XXXX

Applications that use this media type: Applications that publish descriptions of URI Interactions.

Additional information:

Magic number(s): none

File extension(s): No specific file extension proposed, but as a general rule, XML data often uses ".xml" as the file extension.

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Erik Wilde <dret@berkeley.edu>

Intended usage: COMMON

Restrictions on usage: none

Author: Erik Wilde <dret@berkeley.edu>

Change controller: IETF

6.2. Link Relation

The link relation type below will be registered by IANA per Section 6.2.1 of RFC 5988 [RFC5988]:

Relation Name: ldesc

Description: Linking to a resource that can be used as a link description for requesting runtime information about a particular context's interaction affordances.

Reference: RFC XXXX

Notes: Link Descriptions can be used in all scenarios where clients want to create requests that represent a query into the context resource. The media type of the context resource and the media type of the link description resource are not constrained by this specification.

7. Implementation Status

Note to RFC Editor: Please remove this section before publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 6982 [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 6982, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

8. Security Considerations

...

9. Open Issues

If and how to use profiles (example in Section 5); if profile use is recommended, define a suggested profile URI for other specs to use?

How to handle variables in Level 4 templates that are supposed to have composite values?

If a template is refined in an incremental process (such as for example faceted search services), does it make sense to be able to add a "back" link and/or "home" link, so that clients can find the "most general" version easily?

How does this interact with "faceted search" scenarios? Does incremental refinement of URI Template Descriptions somehow nicely and naturally map into faceted search scenarios?

Is there a concept of how Template Descriptions (and thus URI Templates) can be reused? Should there be an inclusion facility or something along those lines? If so, what's the model for that? Initial thoughts on possibilities can be found on this page [3]

Should there be some recommended link relation to use when linking to a Template Description from within the context of a URI Template?

While currently everything is defined in XML, providing alternative serializations (JSON and RDF) might be an interesting thing to consider.

10. Change Log

Note to RFC Editor: Please remove this section before publication.

10.1. Prior to -00

An earlier variation of a similar idea was published as "Template Descriptions" [I-D.wilde-template-desc]. However, since this earlier draft was exclusively focusing on interactions with links driven by URI Templates [RFC6570], instead of looking at links in general, it was sufficiently distinct to start a new draft, instead of evolving

the existing one.

11. References

11.1. Normative References

[I-D.nottingham-link-hint]
Nottingham, M., "HTTP Link Hints", draft-nottingham-link-hint-00 (work in progress), June 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

[XSD-1] Thompson, H., Beech, D., Mendelsohn, N., and M. Maloney, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[XSD-2] Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

11.2. Non-Normative References

[I-D.wilde-template-desc]
Wilde, E., Davis, C., and Y. Liu, "URI Template Descriptions", draft-wilde-template-desc-00 (work in progress), December 2012.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5005] Nottingham, M., "Feed Paging and Archiving", RFC 5005, September 2007.

[RFC5023] Gregorio, J. and B. de hOra, "The Atom Publishing Protocol", RFC 5023, October 2007.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.

[RFC6839] Hansen, T. and A. Melnikov, "Additional Media Type Structured Syntax Suffixes", RFC 6839, January 2013.

[RFC6906] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, March 2013.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

URIs

[1] <https://www.ietf.org/mailman/listinfo/apps-discuss>

[2] <https://github.com/dret/I-D/tree/master/link-desc>

[3] <http://dret.typepad.com/dretblog/2012/12/structuring-uri-templates.html>

Appendix A. Acknowledgements

Thanks for comments and suggestions provided by Dmitry Limonov.

Appendix B. Link Description Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="urn:ietf:rfc:XXXX"
xmlns:ld="urn:ietf:rfc:XXXX">
    <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd">
        <xs:annotation>
            <xs:documentation>Get access to the xml: attribute groups for
xml:lang as declared on 'documentation' below.</xs:documentation>
        </xs:annotation>
    </xs:import>
    <xs:element name="link" type="ld:link-type"> </xs:element>
    <xs:simpleType name="uriTemplate">
        <xs:annotation>
```

```
            <xs:documentation>Representing the type for URI template values
according to RFC 6570. This is probably too complicated to cover with a regular
expression in any reasonable way, so type enforcement is not done by the
schema.</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string"/>
```

```
    </xs:simpleType>
    <xs:group name="descriptions">
        <xs:choice>
            <xs:element name="appinfo" id="appinfo">
                <xs:complexType mixed="true">
                    <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:any processContents="lax"/>
                    </xs:sequence>
                    <xs:attribute name="source" type="xs:anyURI"/>
                    <xs:anyAttribute namespace="##other" processContents="lax"/
>
                </xs:complexType>
            </xs:element>
            <xs:element name="documentation" id="documentation">
                <xs:complexType mixed="true">
                    <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:any processContents="lax"/>
                    </xs:sequence>
                    <xs:attribute name="source" type="xs:anyURI"/>
                    <xs:attribute ref="xml:lang"/>
                    <xs:anyAttribute namespace="##other" processContents="lax"/
>
                </xs:complexType>
            </xs:element>
        </xs:choice>
    </xs:group>
    <!-- these are simplified versions of the types for facet elements as
defined in the schema for schemas at http://www.w3.org/2001/XMLSchema.xsd -->
    <xs:complexType name="facet">
        <xs:complexContent>
            <xs:restriction base="xs:anyType">
                <xs:attribute name="value" use="required"/>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="numFacet">
        <xs:complexContent>
            <xs:restriction base="ld:facet">
                <xs:attribute name="value" type="xs:nonNegativeInteger"
use="required"/>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="link-type">
        <xs:choice maxOccurs="unbounded">
            <xs:element name="var">
                <xs:complexType>
                    <xs:sequence maxOccurs="unbounded" minOccurs="1">
```

```
                        <xs:element name="restriction" minOccurs="0">
                            <xs:complexType>
                                <xs:choice maxOccurs="unbounded" minOccurs="0">
                                    <xs:element name="minExclusive"
type="ld:facet"/>
```

```
                                    <xs:element name="minInclusive"
type="ld:facet"/>
                                    <xs:element name="maxExclusive"
type="ld:facet"/>
                                    <xs:element name="maxInclusive"
type="ld:facet"/>
                                    <xs:element name="totalDigits">
                                        <xs:complexType>
                                            <xs:complexContent>
                                                <xs:restriction
base="ld:numFacet">
                                                    <xs:attribute name="value"
type="xs:positiveInteger" use="required"/>
                                                </xs:restriction>
                                            </xs:complexContent>
                                        </xs:complexType>
                                    </xs:element>
                                    <xs:element name="fractionDigits"
type="ld:numFacet"/>
                                    <xs:element name="length"
type="ld:numFacet"/>
                                    <xs:element name="minLength"
type="ld:numFacet"/>
                                    <xs:element name="maxLength"
type="ld:numFacet"/>
                                    <xs:element name="enumeration"
type="ld:facet"/>
                                    <xs:element name="whiteSpace">
                                        <xs:complexType>
                                            <xs:complexContent>
                                                <xs:restriction
base="ld:facet">
                                                    <xs:attribute name="value"
use="required">
                                                        <xs:simpleType>
                                                            <xs:restriction
base="xs:NMTOKEN">
                                                                <xs:enumeration
value="preserve"/>
                                                                <xs:enumeration
value="replace"/>
                                                                <xs:enumeration
value="collapse"/>
                                                            </xs:restriction>
                                                        </xs:simpleType>
                                                    </xs:attribute>
                                                </xs:restriction>
                                            </xs:complexContent>
                                        </xs:complexType>
                                    </xs:element>
                                    <xs:element name="pattern" type="ld:facet"/
>
                                </xs:choice>
                                <xs:attribute name="base" use="required"
type="xs:QName"> </xs:attribute>
                            </xs:complexType>
                        </xs:element>
                        <xs:group ref="ld:descriptions" maxOccurs="unbounded"
minOccurs="0"/>
                        <xs:any namespace="##other" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>If variables are restricted
in ways other than the simple type restrictions that are built into link
descriptions, then these restrictions can be embedded in a variable description
as well, as long as they are represented using a different namespace.</
xs:documentation>
                            </xs:annotation>
                        </xs:any>
                    </xs:sequence>
                    <xs:attribute name="name" use="required"/>
                    <xs:attribute name="concept" type="xs:anyURI"
use="optional">
```

```
                        <xs:annotation>
                            <xs:documentation>Identifies the variable by
referring to a concept URI.</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="default" type="xs:anySimpleType"
use="optional">
                        <xs:annotation>
                            <xs:documentation>Defines the default value for the
variable (used by the service if no value is provided). The value must match
the type defined for the variable.</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
            <xs:group ref="ld:descriptions" maxOccurs="unbounded"
minOccurs="0"/>
            <xs:element name="hint">
                <xs:complexType>
                    <xs:sequence>
                        <xs:group maxOccurs="unbounded" minOccurs="0"
ref="ld:descriptions"/>
                    </xs:sequence>
                    <xs:attribute name="name" use="required">
                        <xs:annotation>
                            <xs:documentation>A hint is either a registered
hint with a simple name (defined by a regular expression), or an unregistered
hint which is identified by URI.</xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                            <xs:union memberTypes="xs:anyURI">
                                <xs:simpleType>
                                    <xs:restriction base="xs:token">
                                        <xs:annotation>
                                            <xs:documentation>The list of
"registered link hints" is taken from http://tools.ietf.org/html/draft-
nottingham-link-hint-00#section-3 and will probably change.</xs:documentation>
                                        </xs:annotation>
                                        <xs:enumeration value="allow"/>
                                        <xs:enumeration value="formats"/>
                                        <xs:enumeration value="links"/>
                                        <xs:enumeration value="accept-post"/>
                                        <xs:enumeration value="accept-patch"/>
                                        <xs:enumeration value="accept-ranges"/>
                                        <xs:enumeration value="accept-prefer"/>
                                        <xs:enumeration value="precondition-
req"/>

                                        <xs:enumeration value="auth-scheme"/>
                                        <xs:enumeration value="status"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:union>
                        </xs:simpleType>
                    </xs:attribute>
                    <xs:attribute name="value"/>
                    <xs:assert test="count(../ld:hint[@name = current()/@name])
&lt;= 1">
                        <xs:annotation>
                            <xs:documentation>It is not allowed to have more
than one hint with the same @name on the same link.</xs:documentation>
                        </xs:annotation>
                    </xs:assert>
                </xs:complexType>
            </xs:element>
        </xs:choice>
        <xs:attribute name="hreft" type="ld:uriTemplate" use="optional"/>
        <xs:attribute name="href" type="xs:anyURI" use="optional"/>
        <xs:assert test="count(@href | @hreft) = 1">
            <xs:annotation>
                <xs:documentation>A link must either specify a URI (@href) or a
URI Template (@hreft), but it cannot specify both at the same time.</
xs:documentation>
            </xs:annotation>
        </xs:assert>
    </xs:complexType>
</xs:schema>
```

Appendix C. XSLT for Generating Link Description HTML

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
xmlns:ld="urn:ietf:rfc:XXXX" exclude-result-prefixes="ld">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>Link Descriptions</title>
                <style type="text/css">
                    ul { margin : 0 }
                    .msg { color : #C0C0C0 }
                </style>
            </head>
            <body>
                <h1>Link Descriptions</h1>
                <xsl:for-each select="//*[ ld:var | ld:hint ]">
                    <hr/>
                    <xsl:for-each select=" @href | @hreft | @ld:href |
@ld:hreft ">
                        <h2>
                            <code>
                                <xsl:value-of select="name()"/>
                                <xsl:text>="</xsl:text>
                                <a href="{.}">
                                    <xsl:value-of select="."/>
                                </a>
                                <xsl:text>"</xsl:text>
                            </code>
                        </h2>
                    </xsl:for-each>
                    <xsl:if test="@rel">
```

```
                                <h3>
                                    <code>
                                        <xsl:text>rel="</xsl:text>
                                        <xsl:value-of select="@rel"/>
                                        <xsl:text>"</xsl:text>
                                    </code>
                                </h3>
                            </xsl:if>
                            <table rules="none" style="margin : 15px">
                                <tr>
                                    <th align="right">Documentation:</th>
                                    <td>
                                        <xsl:call-template name="documentation"/>
                                    </td>
                                </tr>
                                <tr>
                                    <th align="right">Appinfo:</th>
                                    <td>
                                        <xsl:call-template name="appinfo"/>
                                    </td>
                                </tr>
                            </table>
                            <xsl:if test="ld:var">
                                <h4>Variables:</h4>
                                <table rules="all" border="1" cellpadding="5">
                                    <thead>
                                        <tr>
                                            <th>Variable</th>
                                            <th>Concept</th>
                                            <th>Default</th>
                                            <th>Value Range</th>
                                            <th>Documentation</th>
                                            <th>Appinfo</th>
                                        </tr>
                                    </thead>
                                    <xsl:for-each select="ld:var">
                                        <xsl:sort select="@name"/>
                                        <tr>
                                            <td>
                                                <xsl:value-of select="@name"/>
                                            </td>
                                            <td>
                                                <xsl:choose>
                                                    <xsl:when test="@concept">
                                                        <q>
                                                            <xsl:value-of
        select="@concept"/>

                                                        </q>
                                                    </xsl:when>
                                                    <xsl:otherwise>
                                                        <span class="msg">n/a</span>
                                                    </xsl:otherwise>
                                        </xsl:choose>
```

```
                                    </td>
                                    <td>
                                        <xsl:choose>
                                            <xsl:when test="@default">
                                                <q>
                                                    <code>
                                                        <xsl:value-of
select="@default"/>
                                                    </code>
                                                </q>
                                            </xsl:when>
                                            <xsl:otherwise>
                                                <span class="msg">n/a</span>
                                            </xsl:otherwise>
                                        </xsl:choose>
                                    </td>
                                    <td>
                                        <xsl:choose>
                                            <xsl:when test="ld:restriction">
                                                <div>
                                                    <xsl:text>Base: </xsl:text>
                                                    <code>
                                                        <xsl:choose>
                                                            <xsl:when
test="contains(ld:restriction/@base, ':')">
                                                                <xsl:value-of
select="ld:restriction/@base"/>
                                                            </xsl:when>
                                                            <xsl:otherwise>
                                                                <a
href="http://www.w3.org/TR/xmlschema-2/#{ld:restriction/@base}">
                                                                    <xsl:value-
of select="concat('xs:', ld:restriction/@base)"/>
                                                                </a>
                                                            </xsl:otherwise>
                                                        </xsl:choose>
                                                    </code>
                                                </div>
                                                <xsl:if test="ld:restriction/
ld:*">
                                                    <ul>
                                                        <xsl:for-each
select="ld:restriction/ld:*">
                                                            <xsl:sort
select="local-name()"/>
                                                            <li>
                                                                <code>
                                                                    <a
```

```
href="http://www.w3.org/TR/xmlschema-2/#rf-{local-name()}">

<xsl:value-of select="concat('xs:',local-name())"/>
                                                                     </a>

<xsl:text>="</xsl:text>
                                                                     <xsl:value-
of select="@value"/>

<xsl:text>"</xsl:text>
                                                                 </code>
                                                             </li>
                                                         </xsl:for-each>
```

```
                                                    </ul>
                                                </xsl:if>
                                            </xsl:when>
                                            <xsl:otherwise>
                                                <span class="msg">n/a</span>
                                            </xsl:otherwise>
                                        </xsl:choose>
                                    </td>
                                    <td>
                                        <xsl:call-template
name="documentation"/>
                                    </td>
                                    <td>
                                        <xsl:call-template name="appinfo"/>
                                    </td>
                                </tr>
                            </xsl:for-each>
                        </table>
                    </xsl:if>
                    <xsl:if test="ld:hint">
                        <h4>Hints:</h4>
                        <table rules="all" border="1" cellpadding="5">
                            <thead>
                                <tr>
                                    <th>Name</th>
                                    <th>Value</th>
                                    <th>Documentation</th>
                                    <th>Appinfo</th>
                                </tr>
                            </thead>
                            <xsl:for-each select="ld:hint">
                                <xsl:sort select="@name"/>
                                <tr>
                                    <td>
                                        <xsl:value-of select="@name"/>
                                    </td>
                                    <td>
                                        <code>
                                            <xsl:value-of select="@value"/>
                                        </code>
                                    </td>
                                    <td>
                                        <xsl:call-template
name="documentation"/>
                                    </td>
                                    <td>
                                        <xsl:call-template name="appinfo"/>
                                    </td>
```

```
                </tr>
            </xsl:for-each>
```

```
                        </table>
                    </xsl:if>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>
    <xsl:template name="documentation">
        <xsl:choose>
            <xsl:when test="ld:documentation">
                <ul>
                    <xsl:for-each select="ld:documentation">
                        <li>
                            <xsl:value-of select="node()"/>
                            <xsl:if test="@xml:lang">
                                <xsl:text> </xsl:text>
                                <span class="msg">
                                    <xsl:text>(</xsl:text>
                                        <code>
                                            <xsl:text>xml:lang="</xsl:text>
                                            <xsl:value-of select="@xml:lang"/>
                                            <xsl:text>"</xsl:text>
                                        </code>
                                    <xsl:text>)</xsl:text>
                                </span>
                            </xsl:if>
                        </li>
                    </xsl:for-each>
                </ul>
            </xsl:when>
            <xsl:otherwise>
                <span class="msg">n/a</span>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
    <xsl:template name="appinfo">
        <xsl:choose>
            <xsl:when test="ld:appinfo">
                <ul>
                    <xsl:for-each select="ld:appinfo">
                        <li>
                            <xsl:text>Source: </xsl:text>
                            <xsl:value-of select="@source"/>
                        </li>
                    </xsl:for-each>
                </ul>
            </xsl:when>
            <xsl:otherwise>
                <span class="msg">n/a</span>
```

```
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
</xsl:stylesheet>
```

Author's Address

Erik Wilde
EMC Corporation
6801 Koll Center Parkway
Pleasanton, CA 94566
U.S.A.

Phone: +1-925-6006244
Email: erik.wilde@emc.com
URI: http://dret.net/netdret/