Link-based Resource Descriptor Discovery
draft-hammer-discovery-03

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 24, 2009.

Copyright Notice

Copyright (c) 2009 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 in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This memo describes LRDD (pronounced ‘lard’), a process for obtaining information about a resource identified by a URI. The ‘information about a resource’, a resource descriptor, provides machine-readable
information that aims to increase interoperability and enhance the interaction with the resource. This memo only defines the process for locating and obtaining the descriptor, but leaves the descriptor format and its interpretation out of scope.

Table of Contents

1. Introduction ................................................. 3
2. Notational Conventions ................................... 4
3. The describedby Link Relation ............................ 4
4. Identifying Descriptor Location ........................... 5
   4.1. Method Selection ..................................... 5
   4.2. The <LINK> Element .................................. 6
   4.3. The HTTP Link Header ................................. 7
   4.4. The Host Metadata Document ......................... 8
5. Obtaining Resource Descriptor .............................. 9
6. The Link-Pattern host-meta Field ......................... 9
   6.1. Template Syntax ..................................... 10
7. Security Considerations .................................... 11
8. IANA Considerations ..................................... 11
   8.1. The Link-Pattern host-meta Field .................. 11
   8.2. The describedby Relation Type ....................... 12
Appendix A. Descriptor Discovery vs. Service Discovery ........ 12
Appendix B. Methods Suitability Analysis .................... 13
Appendix B.1. Requirements ................................. 13
Appendix B.2. Analysis ....................................... 15
Appendix C. Acknowledgments .................................. 22
Appendix D. Document History ................................. 22
9. References ................................................. 24
   9.1. Normative References ............................... 24
   9.2. Informative References ............................ 25
Author’s Address ............................................ 25
1. Introduction

This memo defines a process for locating descriptors for resources identified with URIs. Resource descriptors are documents (usually based on well known serialization languages such as XML, RDF, and JSON) which provide machine-readable information about resources (resource metadata) for the purpose of promoting interoperability and assist in interacting with unknown resources that support known interfaces.

While many methods provide the ability to link a resource to its metadata, none of these methods fully address the requirements of a uniform and easily implementable process. These requirements include the ability for resources to self-declare the location of their descriptors, the ability to access descriptors directly without interacting with the resource, and support a wide range of platforms and scale of deployment. They must also be fully compliant with existing web protocols, and support extensibility. These requirements, and the analysis used as the basis for this memo are explains in detail in Appendix B.

For example, a web page about an upcoming meeting can provide in its descriptor document the location of the meeting organizer’s free/busy information to potentially negotiate a different time. A social network profile page descriptor can identify the location of the user’s address book as well as accounts on other sites. A web service implementing an API with optional components can advertise which of these are supported.

This memo describes the first step in the discovery process in which the resource descriptor document is located and retrieved. Other steps, which are outside the scope of this memo, include parsing the descriptor document based on its format (such as POWDER [POWDER], XRD [XRD], and Metalink [I-D.bryan-metalink]) and utilizing it based on the application.

Discovery can be performed before, after, or without obtaining a representation of the resource. Performing discovery ahead of accessing a representation allows the client not to reply on assumptions about the properties of the resource. Performing discovery after a representation has been obtained enables further interaction with it.

Given the wide range of ‘information about a resource’, no single descriptor format can adequately accommodate such scope. However, there is great value in making the process locating the descriptor uniform across formats. While HTTP is the most common protocol used in association with discovery and is explicitly specified in this
memo, other protocols MAY be used.

Please discuss this draft on the www-talk@w3.org [1] mailing list.

2. Notational Conventions

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 [RFC2119].

This document uses the Augmented Backus-Naur Form (ABNF) notation of [RFC2616]. Additionally, the following rules are included from [RFC3986]: reserved and unreserved, and from [I-D.nottingham-http-link-header]: link-param.

3. The describedby Link Relation

The methods described in this memo express the location of the resource descriptor as a link relation, utilizing the link framework defined by [I-D.nottingham-http-link-header]. The association of a descriptor document with the resource it describes is declared using the "describedby" link relation type.

The "describedby" link relation is defined in [POWDER] and registered as:

   The relationship A "describedby" B asserts that resource B provides a description of resource A. There are no constraints on the format or representation of either A or B, neither are there any further constraints on either resource.

Since a single resource can have many descriptors, the "describedby" link relation has a one-to-many structure (the question whether a single descriptor can describe multiple resources is outside the scope of this memo). In the case of multiple "describedby" links obtained from a single method, selecting which link to use is application-specific.

To promote interoperability, applications referencing this memo SHOULD clearly define the application-specific criteria used to select between "describedby" links. This MAY be done by:

   o Supporting a single descriptor format, or defining an order of precedence for multiple descriptor formats. Applications MAY require the presence of the link "type" attribute with the mime-type of the required format.
Using the "describedby" relation type together with another application-specific relation type in the same link. The application-specific relation type can be registered or an extension.

Specifying additional link attributes using link-extensions.

Link selection MUST NOT depend on the order in which multiple links are obtained from a single method. Applications MUST NOT impose constraints on the usage of the "describedby" relation type as it is likely to be used by other applications in association with the same resource.

4. Identifying Descriptor Location

The descriptor location (URI) is a function of the resource URI. This section defines three methods which together satisfy the requirements defined in Appendix B. While each method on its own satisfies the requirements partially, together they provide enough flexibility for most use cases. Each of the following three methods is performed by using the resource URI to identify its descriptor URI.

In many cases, a request for one URI leads to requesting other URIs, as is the case with HTTP redirections. Because the decision whether to use such URIs is application-specific, discovery is constrained to a single URI identifying the resource. Any other resource URIs received MUST be considered as a separate and discrete input into the discovery function. If a resource URI obtained during the performance of these methods is found to be more relevant to the application, the discovery process MUST be restarted with the new resource URI as its input.

For example, an HTTP HEAD request for URI A returns a redirect (307) response with a set of "describedby" links, and identifies the temporary location of the representation at URI B. An HTTP HEAD request for URI B returns a successful (200) response with its own set of "describedby" links. An application MAY choose to define a process in which the two sets of links are obtained, prioritized, and utilized, however, it MUST do so by explicitly instructing the client to perform discovery multiple times, as each is considered separate and distinct discovery.

4.1. Method Selection

Each method presents a different set of requirements. The criteria used to determine which methods a server SHOULD support and client
SHOULD attempt are based on a combination of factors:

- The ability to offer and obtain a representation of the resource by dereferencing its URI.

- The availability of a representation supporting <LINK> markup compatible with [I-D.nottingham-http-link-header].

- The availability of an HTTP representation of the resource and the ability to provide and access link information in its response header.

The methods are listed based on the restrictiveness of their requirements in descending order, from the most specialized to the most generic. This ordering, however, does not imply the order in which multiple applicable methods should be attempted. Because different methods are more appropriate in different circumstances, it is up to each application to define how they should be used together.

To promote interoperability, applications referencing this memo MUST clearly define the relationship between the three methods as either:

- equal, all methods MUST produce the same set of resource descriptors and clients MAY attempt either method according to their capabilities, or

- with an application-specific order of precedence, where methods MUST be attempted in a specific order.

4.2. The <LINK> Element

The <LINK> element method is limited to resources with an available markup representation that supports typed-relations using the <LINK> element, such as HTML [W3C.REC-html401-19991224], XHTML [W3C.REC-xhtml1-20020801], and Atom [RFC4287]. Other markup formats are permitted as long as the semantics of their <LINK> elements are fully compatible with the link framework defined in [I-D.nottingham-http-link-header]. This method requires the retrieval of a resource representation. While HTTP is the most common transport for such documents, this method is transport independent.

For example:

```html
<LINK href="http://example.com/resource;about"
     rel="describedby" type="application/powder+xml">
```

A client trying to obtain the location of the resource’s descriptor
using this method SHALL:

1. Retrieve a representation of the resource using the applicable transport for that resource URI. If the markup document is obtained using HTTP, it MUST only be used by the client if the document is a valid representation of the resource identified by the HTTP request URI, typically in a response with a successful (2xx) or redirection (3xx) status code. If no such valid representation of the request URI is found, the method fails.

2. Parse the document as defined by its format specification and look for <LINK> elements with a "rel" attribute value containing the "describedby" relation. The client MUST obey the document markup schema and ignore any invalid elements (such as <LINK> elements outside the <HEAD> section of an HTML document). This is done to avoid unintentional markup from other parts of the document to be used for discovery purposes, which can have vast impact on usability and security.

3. Narrow down the selection if more than one "describedby" link is found, following the application-specific criteria. The descriptor location is obtained from the value of the "href" attribute in the selected <LINK> element.

   <LINK> elements MAY include other relation types together with "describedby" in a single "rel" attribute (for example 'rel="describedby copyright"'). Clients MUST be properly process use such multiple relation "rel" attributes as defined by the format specification.

4.3. The HTTP Link Header

The HTTP Link header method is limited to resources for which an HTTP GET or HEAD request returns a 2xx, 3xx, or 4xx HTTP response [RFC2616]. This method uses the Link header defined in [I-D.nottingham-http-link-header] and requires the retrieval of a resource representation header.

For example:

   Link: <http://example.com/resource;about>; rel="describedby"; type="application/powder+xml"

A client trying to obtain the location of the resource’s descriptor using this method SHALL:

1. Make an HTTP (or HTTPS as required) GET or HEAD request to the resource URI to obtain a valid response header. If the HTTP
response carries a status code other than successful (2xx), redirection (3xx), or client error (4xx), the method fails.

2. Parse the HTTP response header and look for Link headers with a "rel" parameter value containing the "describedby" relation.

3. Narrow down the selection if more than one "describedby" link is found, following the application-specific criteria. The descriptor location is obtained from the "< >" enclosed URI-reference in the selected Link header.

Link headers MAY include other relation types together with "describedby" in a single "rel" parameter (for example 'rel="describedby copyright"'). Clients MUST be properly process use such multiple relation "rel" attributes as defined by [I-D.nottingham-http-link-header].

4.4. The Host Metadata Document

The host metadata document method is available for any resource identified by a URI whose authority supports the host-meta document defined in [I-D.nottingham-site-meta]. This method does not require obtaining any representation of the resource, and operates solely using the resource URI.

The link relation between the resource URI and the descriptor URI is obtained by using a template contained in the host-meta document. By applying the host-wide template to an individual resource URI, a resource-specific link is produced which can be used to indicate the location of the descriptor document for that resource, bypassing the need to access or provide a representation for it.

For example (line breaks are for formatting only, and are not allowed in the document):

    Link-Pattern: <{uri};about">; rel="describedby";
                   type="application/powder+xml"

A client trying to obtain the location of the resource’s descriptor using this method SHALL:

1. Retrieve the host-meta document for URI’s authority as defined by [I-D.nottingham-site-meta] section 4. If the request fails to retrieve a valid host-meta document, the method fails.

2. Parse host-meta document and look for Link-Pattern fields with a "rel" attribute value containing the "describedby" relation.
3. Narrow down the selection if more than one "describedby" link is found, following the application-specific criteria. The descriptor location is constructed by applying the template obtained from the selected Link-Pattern field to the resource URI as described by Section 6.1.

Link-Pattern MAY include other relation types together with "describedby" in a single "rel" parameter (for example 'rel="describedby copyright"'). Clients MUST be properly process use such multiple relation "rel" attributes as defined by Section 6.

5. Obtaining Resource Descriptor

Once the desired descriptor URI has been obtained, the descriptor document is retrieved. If the descriptor URI scheme is "http" or "https", the document is obtained via an HTTP (or HTTPS as required) GET request to the identified URI. The client MUST obey HTTP redirections (3xx), and the descriptor document is considered valid only if retrieved with a successful HTTP response status (2xx).

6. The Link-Pattern host-meta Field

The Link host-meta field [I-D.nottingham-site-meta] conveys a link relation between all resource URIs under the host-meta authority and a common target URI. However, there are cases in which relations of different resources with the same authority do not share the same target URI, but do follow a common pattern in how the target URI is constructed.

For example, a news site with multiple authors can provide information about each article’s author, but appending a suffix (such as ";by") to the URI of each article. Each article has a unique author, but all share the same pattern of where that information is located. The same information can be provided using an HTTP link header or HTML <LINK> element, but in a less efficient manner when a single pattern can provide the same information:

   Link-Pattern: <{uri};by>; rel="author"

The Link-Pattern host-meta field uses a slightly modified syntax of the HTTP Link header [I-D.nottingham-http-link-header] to convey relations whose context is individual resources with the same authority as the host-meta document, and whose target is constructed by applying a template to the context URI. The field is not specific to any relation type and MAY be used to express any relations supported by the Link header [I-D.nottingham-http-link-header].
The Link-Pattern host-meta field differs from the HTTP Link header in the following respects:

- The "<>" enclosed token is not a valid URI, but instead contains a template as defined in Section 6.1.
- Its context URI is defined as the individual resource URI used as input to the template.
- If the resulting target URI expressed by the template is relative, its base URI is the root resource of the authority.

```
Link-Pattern = "Link-Pattern" ":" #pattern-value
pattern-value = "<" template ">">" *( ";" link-param )
template = *( uri-char | "{" [ ";" ] var-name "}" )
uri-char = ( reserved | unreserved )
var-name = "scheme" | "authority" | "path" | "query" | "fragment" | "userinfo" | "host" | "port" | "uri"
```

[[ should this spec define a filter/map parameter that will allow applying link patterns to subsets of the host-meta scope? This can use a regular expression match or something similar to robots.txt. If the spec will end up not directly supporting this feature, I will add a note suggesting that such a feature could be defined elsewhere as an extension. ]]

6.1. Template Syntax

The template syntax provides a simple format for URI transformation. A template is a string containing brace-enclosed ("{}") variable names marking the parts of the string that are to be substituted by the variable values. A template is transformed into a URI by substituting the variables with their calculated value. If a variable name is prefixed by ";", any character in the variable value other than unreserved MUST be percent-encoded per [RFC3986].

To construct a URI using a template, the input URI is parsed into its URI components and each component value assigned to a variable name. The template variable substitution is based on the URI vocabulary defined by [RFC3986] section 3 and includes: "scheme", "authority", "path", "query", "fragment", "userinfo", "host", and "port". In addition, it defines the "uri" variable as the entire input URI
excluding the fragment component and the "#" fragment separator.

```
foo://william@example.com:8080/over/there?name=ferret#nose
```

For example, given the input URI "http://example.com/r/1?f=xml#top",
each of the following templates will produce the associated output
URI:

```
http://example.org?q={%uri} -->
http://example.org?q=http%3A%2F%2Fexample.com%2Fr%2F1%3Ff%3Dxml
```

```
http://meta.{host}:8080{path}?{query} -->
http://meta.example.com:8080/r/1?f=xml
```

```
https://{authority}/v1{path}#{fragment} -->
https://example.com/v1/r/1#top
```

7. Security Considerations

The methods used to perform discovery are not secure, private or
integrity-guaranteed, and due caution should be exercised when using
them. Applications that perform discovery should consider the attack
vectors opened by automatically following, trusting, or otherwise
using links gathered from <LINK> elements, HTTP Link headers, or
host-meta documents.

8. IANA Considerations

8.1. The Link-Pattern host-meta Field

This specification registers the Link-Pattern host-meta field in the
host-meta Field Registry [I-D.nottingham-site-meta].
Field Name: Link-Pattern

Change controller: IETF

Specification document(s): [[ this document ]]

Related information: [I-D.nottingham-http-link-header]

8.2. The describedby Relation Type

[[ this section will be removed if the "describedby" relation type is registered by the time it is published ]]

This specification registers the "describedby" relation type in the Link Relation Type Registry [I-D.nottingham-http-link-header].

- Relation Name: describedby
- Description: The relationship A "describedby" B asserts that resource B provides a description of resource A. There are no constraints on the format or representation of either A or B, neither are there any further constraints on either resource.
- Documentation: [POWDER]

Appendix A. Descriptor Discovery vs. Service Discovery

Descriptor discovery provides a process for obtaining information about a resource identified with a URI. It allows servers to describe their resources in a machine-readable format, enabling automatic interoperability by user-agents and resource consuming applications. Discovery enables applications to utilize a wide range of web services and resources across multiple providers without the need to know about their capabilities in advance, reducing the need for manual configuration and resource-specific software.

When discussing discovery, it is important to differentiate between descriptor discovery and service discovery. Both types attempt to associate capabilities with resources, but they approach it from opposite ends.

Service discovery centers on identifying the location of qualified resources, typically finding an endpoint capable of certain protocols and capabilities. In contrast, descriptor discovery begins with a resource, trying to find which capabilities it supports.

A simple way to distinguish between the two types of discovery is to
define the questions they are each trying to answer:

Descriptor-Discovery: Given a resource, what are its attributes: capabilities, characteristics, and relationships to other resources?

Service-Discovery: Given a set of attributes, which available resources match the desired set and what is their location?

While this memo deals exclusively with descriptor discovery, it is important to note that the two discovery types are closely related and are usually used in tandem. In fact, a typical use case will switch between service discovery and descriptor discovery multiple times in a single workflow, and can start with either one.

One reason for this dependency between the two discovery types is that resource descriptors usually contain not only a list of capabilities, but also relationships to other resources. Since those relationships are usually typed, the process in which an application chooses which links to use is in fact service discovery.

Applications use descriptor discovery to obtain the list of links, and service discovery to choose the relevant links. In another common example, the application uses service discovery to find a resource with a given capability, then uses descriptor discovery to find out what other capabilities it supports.

Appendix B. Methods Suitability Analysis

Due to the wide range of use cases requiring resource descriptors, and the desire to reuse as much as possible, no single solution has been found to sufficiently cover the requirements for linking between the resource URI and the descriptor URI. The following analysis attempts to list all the method proposed for addressing descriptor discovery. It is included here to provide background information as to why certain methods have been selected while others rejected from the discovery process. It has been updated to match the terms used in this memo and its structure.

Appendix B.1. Requirements

Getting from a resource URI to its descriptor document can be implemented in many ways. The problem is that none of the current methods address all of the requirements presented by the common use cases. The requirements are simple, but the more we try to address, the less elegant and accessible the process becomes. While working on the now defunct XRDS-Simple specification [XRDS-Simple] and
talking to companies and individual about it, the following requirements emerged for any proposed process:

Self Declaration:

Allow resources to declare the availability of descriptor information and its location. When a resource is accessed, it needs to have a way to communicate to the client that it supports the discovery protocol and to indicates the location of such descriptor.

This is useful when the client is able or is already interacting with the resource but can enhance its interaction with additional information. For example, accessing a blog page enhanced if it was generated from an Atom feed or Atom entry and that feed supports Atom authoring.

Direct Descriptor Access:

Enable direct retrieval of the resource descriptor without interacting with the resource itself. Before a resource is accessed, the client should have a way to obtain the resource descriptor without accessing the resource. This is important for two reasons.

First, accessing an unknown resource may have undesirable consequences. After all, the information contained in the descriptor is supposed to inform the client how to interact with the resource. The second is efficiency - removing the need to first obtain the resource in order to get its descriptor (reducing HTTP round-trips, network bandwidth, and application latency).

Web Architecture Compliant:

Work with well-established web infrastructure. This may sound obvious but it is in fact the most complex requirement. Deploying new extensions to the HTTP protocol is a complicated endeavor. Beside getting applications to support a new header, method, or content negotiation, existing caches and proxies must be enhanced to properly handle these requests, and they must not fail performing their normal duties without such enhancements.

For example, a new content negotiation method may cause an existing cache to serve the wrong data to a non-discovery client due to its inability to distinguish the metadata request from the resource representation request.
Scale and Technology Agnostic:

Support large and small web providers regardless of the size of operations and deployment. Any solution must work for a small hosted web site as well as the world largest search engine. It must be flexible enough to allow developers with restricted access to the full HTTP protocol (such as limited access to request or response headers) to be able to both provide and consume resource descriptors. Any solution should also support caching as much as possible and allow reuse of source code and data.

Extensible:

Accommodate future enhancements and unknown descriptor formats. It should support the existing set of descriptor formats such as XRD and POWDER, as well as new descriptor relationships that might emerge in the future. In addition, the solution should not depend on the descriptor format itself and work equally well with any document format - it should aim to keep the road and destination separate.

Appendix B.2. Analysis

The following is a list of proposed and implemented methods trying to address descriptor discovery. Each method is reviewed for its compliance with the requirements identified previously. The [−], [+], or [+-] symbols next to each requirement indicate how well the method complies with the requirement.

Appendix B.2.1. HTTP Response Header

When a resource representation is retrieved using an HTTP GET request, the server includes in the response a header pointing to the location of the descriptor document. For example, POWDER uses the "Link" response header to create an association between the resource and its descriptor. XRDS [XRDS] (based on the Yadis protocol [Yadis]) uses a similar approach, but since the Link header was not available when Yadis was first drafted, it defines a custom header X-XRDS-Location which serves a similar but less generic purpose.

[+] Self Declaration - using the Link header, any resource can point to its descriptor documents.

[−] Direct Descriptor Access - the header is only accessible when requesting the resource itself via an HTTP GET request. While HTTP GET is meant to be a safe operation, it is still possible for some resource to have side-effects.
[+] Web Architecture Compliant – uses the Link header which is an IETF Internet Standard [[ currently a standard-track draft ]], and is consistent with HTTP protocol design.

[-] Scale and Technology Agnostic – since discovery accounts for a small percent of resource requests, the extra Link header is wasteful. For some hosted servers, access to HTTP headers is limited and will prevent implementation.

[+] Extensible – the Link header provides built-in extensibility by allowing new link relations, mime-types, and other extensions.

Minimum roundtrips to retrieve the resource descriptor: 2

Appendix B.2.2. HTTP Response Header Via HEAD

Same as the HTTP Response Header method but used with an HTTP HEAD request. The idea of using the HEAD method is to solve the wasteful overhead of including the Link header in every reply. By limiting the appearance of the Link header only to HEAD responses, typical GET requests are not encumbered by the extra bytes.

[+] Self Declaration – Same as the HTTP Response Header method.

[-] Direct Descriptor Access – Same as the HTTP Response Header method.

[-] Web Architecture Compliant – HTTP HEAD should return the exact same response as HTTP GET with the sole exception that the response body is omitted. By adding headers only to the HEAD response, this solution violates the HTTP protocol and might not work properly with proxies as they can return the header of the cached GET request.

[+] Scale and Technology Agnostic – solves the wasted bandwidth associated with the HTTP Response Header method, but still suffers from the limitation imposed by requiring access to HTTP headers.

[+] Extensible – Same as the HTTP Response Header method.

Minimum roundtrips to retrieve the resource descriptor: 2

Appendix B.2.3. HTTP Content Negotiation

Using the HTTP Accept request header or Transparent Content Negotiation as defined in [RFC2295], the client informs the server it is interested in the descriptor and not the resource itself, to which the server responds with the descriptor document or its location. In
Yadis, the client sends an HTTP GET (or HEAD) request to the resource URI with an Accept header and content-type application/xrds+xml. This informs the server of the client’s discovery interest, which in turn may reply with the descriptor document itself, redirect to it, or return its location via the X-XRDS-Location response header.

[-] Self Declaration - does not address as it focuses on the client declaring its intentions.

[+] Direct Descriptor Access - provides a simple method for directly requesting the descriptor document.

[-] Web Architecture Compliant - while it can be argued that the descriptor can be considered another representation of the resource, it is very much external to it. Using the Accept header to request a separate resource (as opposed to a different representation of the same resource) violates web architecture. It also prevents using the discovery content-type as a valid (self-standing) web resource having its own descriptor.

[-] Scale and Technology Agnostic - requires access to HTTP request and response headers, as well as the registration of multiple handlers for the same resource URI based on the Accept header. In addition, improper use or implementation of the Vary header in conjunction with the Accept header will cause caches to serve the descriptor document instead of the resource itself - a great concern to large providers with frequently visited front-pages.

[-] Extensible - applies an implicit relation type to the descriptor mime-type, limiting descriptor formats to a single purpose. It also prevents using existing mime-types from being used as a descriptor format.

Minimum roundtrips to retrieve the resource descriptor: 1

Appendix B.2.4. HTTP Header Negotiation

Similar to the HTTP Content Negotiation method, this solution uses a custom HTTP request header to inform the server of the client’s discovery intentions. The server responds by serving the same resource representation (via an HTTP GET or HEAD requests) with the relevant Link headers. It attempts to solve the HTTP Response Header waste issue by allowing the client to explicitly request the inclusion of Link headers. One such header can be called "Request-links" to inform the server the client would like it to include certain Link headers of a given "rel" type in its reply.
[+] Self Declaration - same as HTTP Response Header with the option of selective inclusion.

[-] Direct Descriptor Access - does not address.

[-] Web Architecture Compliant - HTTP does not include any mechanism for header negotiation and any custom solution will break existing caches.

[+-] Scale and Technology Agnostic - Requires advance access to HTTP headers on both the client and server sides, but solves the bandwidth waste issue of the HTTP Response Header method.

[+] Extensible - builds on top of Link header extensibility.

Minimum roundtrips to retrieve the resource descriptor: 2

Appendix B.2.5. <Link> Element

Embeds the location of the descriptor document within the resource representation by leveraging the HTML <Link> header element (as opposed to the HTTP header). Applies to HTML resource representations or similar markup-based formats with support for "Link"-like elements such as Atom. POWDER uses the <Link> element in this manner, while XRDS uses the HTML <meta> element with an "http-equiv" attribute equals to X-XRDS-Location (to create an embedded version of the X-XRDS-Location custom header).

[+] Self Declaration - similar to HTTP Response Header method but limited to HTML resources.

[-] Direct Descriptor Access - the method requires fetching the entire resource representation in order to obtain the descriptor location. In addition, it requires changing the resource HTML representation which makes discovery an intrusive process.

[+] Web Architecture Compliant - uses the <Link> element as designed.

[+] Scale and Technology Agnostic - while this solution requires direct retrieval of the resource and manipulation of its content, it is extremely accessible in many platforms.

[-] Extensible - extensibility is restricted to HTML representations or similar markup formats with support for a similar element.

Minimum roundtrips to retrieve the resource descriptor: 2
Appendix B.2.6. HTTP OPTIONS Method

The HTTP OPTIONS method is used to interact with the HTTP server with regard to its capabilities and communication-related information about its resources. The OPTIONS method, together with an optional request header, can be used to request both the descriptor location and descriptor content itself.

[-] Self Declaration - does not address.

[+] Direct Descriptor Access - provides a clean mechanism for requesting descriptor information about a resource without interacting with it.

[+] Web Architecture Compliant - uses an existing HTTP featured.

[-] Scale and Technology Agnostic - requires client and server access to the OPTIONS HTTP method. Also does not support caching which makes this solution inefficient.

[+] Extensible - built-into the OPTIONS method.

Minimum roundtrips to retrieve the resource descriptor: 1

Appendix B.2.7. WebDAV PROPFIND Method

Similar to the HTTP OPTIONS method, the WebDAV PROPFIND method defined in [RFC4918] can be used to request resource specific properties, one of which can hold the location of the descriptor document. PROPFIND, unlike OPTIONS, cannot return the descriptor itself, unless it is returned in the required PROPFIND schema (a multi-status XML element). Other alternatives include URIQA [URIQA], an HTTP extension which defines a method called MGET, and ARK (Archival Resource Key) [ARK] - a method similar to PROPFIND that allows the retrieval of resource attributes using keys (which describe the resource).

[-] Self Declaration - does not address.

[+] Direct Descriptor Access - does not require interaction with the resource, but does require at least two requests to get the descriptor (get location, get document).

[+] Web Architecture Compliant - uses an HTTP extension with less support than core HTTP, but still based on published standards.
[-] Scale and Technology Agnostic - same as the HTTP OPTIONS Method.

[+] Extensible - uses extensible protocols but at the same time depends on solutions that have already gone beyond the standard HTTP protocol, which makes further extensions more complex and unsupported.

Minimum roundtrips to retrieve the resource descriptor: 2

Appendix B.2.8. Custom HTTP Method

Similar to the HTTP OPTIONS Method, a new method can be defined (such as DISCOVER) to return (or redirect to) the descriptor document. The new method can allow caching.

[-] Self Declaration - does not address.

[+] Direct Descriptor Access - same as the HTTP OPTIONS Method.

[-] Web Architecture Compliant - depends heavily on extending every platform to support the extension. Unlikely to be supported by existing proxy services and caches.

[-] Scale and Technology Agnostic - same as HTTP OPTIONS Method with the additional burden on smaller sites requiring access to the new protocol.

[+] Extensible - new protocol that can extend as needed.

Minimum roundtrips to retrieve the resource descriptor: 1

Appendix B.2.9. Static Resource URI Transformation

Instead of using HTTP facilities to access the descriptor location, this method defines a template to transform any resource URI to the descriptor document URI. This can be done by adding a prefix or suffix to the resource URI, which turns it into a new resource URI. The new URI points to the descriptor document. For example, to fetch the descriptor document for http://example.com/resource, the client makes an HTTP GET request to http://example.com/resource;about using a static template that adds the ";about" suffix.

[-] Self Declaration - does not address.

[+] Direct Descriptor Access - creates a unique URI for the descriptor document.
[+] Web Architecture Compliant - uses basic HTTP facilities but intrudes on the domain authority namespace as it defines a static template for URI transformation that is not likely to be compatible with many existing URI naming conventions.

[+] Scale and Technology Agnostic - depending on the static mapping chosen. Some hosted environment will have a problem gaining access to the mapped URI based on the URI format chosen.

[-] Extensible - provides a very specific and limited method to map between resources and their descriptor, since each relation type must mint its own static template.

Minimum roundtrips to retrieve the resource descriptor: 1

Appendix B.2.10. Dynamic Resource URI Transformation

Same as the Static Resource URI Transformation method but with the ability for each domain authority to specify its own discovery transformation template. This can done by placing a configuration file at a known location (such as robots.txt) which contains the template needed to perform the URL mapping. The client first obtains the configuration document (which may be cached using normal HTTP facilities), parses it, then uses that information to transform the resource URI and access the descriptor document.

[+] Self Declaration - does not address individual resources, but allows entire domains to declare their support (and how to use it).

[+] Direct Descriptor Access - once the mapping template has been obtained, descriptors can be accessed directly.

[+] Web Architecture Compliant - uses an existing known-location design pattern (such as robots.txt) and standard HTTP facilities. The use of a known-location if not ideal and is considered a violation of web architecture but if it serves as the last of its kind, can be tolerated. An alternative to the known-location approach can be using DNS to store either the location of the mapping or the map template itself, but DNS adds a layer of complexity not always available.

[+] Scale and Technology Agnostic - works well at the URI authority level (domain) but is inefficient at the URI path level (resource path) and harder to implement when different paths within the same domain need to use different templates. With the decreasing cost of custom domains and sub-domains hosting, this will not be an issue for most services, but it does require sharing configuration
at the domain/sub-domain level.

[+-] Extensible – can be, depending on the schema used to format the known-location configuration document.

Minimum roundtrips to retrieve the resource descriptor: initially 2, 1 after caching

Appendix C. Acknowledgments

With the exception of the host-meta template extension, very little of this memo is original work. Many communities and individuals have been working on solving discovery for many years and this work is a direct result of their hard and dedicated efforts.

Inspiration for this memo derived from previous work on a descriptor format called XRDS-Simple, which in turn derived from another descriptor format, XRDS. Previous discovery workflows include Yadis which is currently used by the OpenID community. While suffering from significant shortcomings, Yadis was a breakthrough approach to performing discovery using extremely restricted hosting environments, and this memo has strived to preserve as much of that spirit as possible.

The use of Link elements and headers and the introduction of the "describedby" relation type in this memo is a direct result of the dedicated work and contribution of Phil Archer to the W3C POWDER specification and Jonathan Rees to the W3C review of Uniform Access to Information About. The host-meta approach was first proposed by Mark Nottingham as an alternative to attaching links directly to resource representations.

The author wishes to thanks the OASIS XRI community for their support, encouragement, and enthusiasm for this work. Special thanks go to Lisa Dusseault, Joseph Holsten, Mark Nottingham, John Panzer, Drummond Reed, and Jonathan Rees for their invaluable feedback.

The author takes all responsibility for errors and omissions.

Appendix D. Document History

[[ to be removed by the RFC editor before publication as an RFC ]]

-03
- Added protocol name LRDD (pronounced ‘lard’).
- Fixed Link-Pattern examples to include missing semicolons.
- Changed focus from an HTTP-based process to Link-based process.
- Completely revised and restructured document for better clarity.
- Realigned the methods to produce consistent results and changed the way redirections and client-errors are handled.
- Updated to use newer version of site-meta, now called host-meta, including a new plaintext-based format to replace the previous XML format.
- Renamed Link-Template to Link-Pattern to avoid future conflict with a previously proposed Link-Template HTTP header.
- Removed support for the "scheme" Link-Template parameter.
- Replaced restrictions with interoperability recommendations.
- Added IANA considerations per new host-meta registry requirements.
- Rename ‘resource discovery’ to ‘descriptor discovery’.
- Added informative reference to Metalink.
- Clarified that the resource descriptor URI can use any URI scheme, not just "http" or "https".
- Removed comment regarding redirects when using <LINK> Elements.
- Clarified that HTTPS must be used with "https" URIs for both Link headers and host-meta retrieval.
- Removed DNS verification step for host-meta with schemes other then "http" and "https". Replaced with a general discussion of authority and a security consideration comment.
- Organized host-meta section into another sub-section level.
- Enlarged the template vocabulary from a single "uri" variable to include smaller URI components.
9. References

9.1. Normative References

[I-D.nottingham-http-link-header]
Nottingham, M., "Link Relations and HTTP Header Linking",
draft-nottingham-http-link-header-03 (work in progress),
November 2008.

[I-D.nottingham-site-meta]
Nottingham, M. and E. Hammer-Lahav, "Host Metadata for the
Web", draft-nottingham-site-meta-01 (work in progress),
February 2009.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

[RFC2295] Holtman, K. and A. Mutz, "Transparent Content Negotiation

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext


Resource Identifier (URI): Generic Syntax", STD 66,


[RFC4918] Dusseault, L., "HTTP Extensions for Web Distributed
Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

[W3C.REC-hmtl401-19991224]
Raggett, D., Jacobs, I., and A. Hors, "HTML 4.01
Specification", World Wide Web Consortium
Recommendation REC-hmtl401-19991224, December 1999,
<http://www.w3.org/TR/1999/REC-hmtl401-19991224>.
9.2. Informative References


URIs

Author’s Address

Eran Hammer-Lahav
Yahoo!

Email: eran@hueniverse.com
URI: http://hueniverse.com