The .well-known/index Mechanism
draft-snell-web-index-00

Abstract

This memo describes an alternative approach to the Web Resource
discovery mechanisms described within the
draft-jones-simple-web-discovery and draft-jones-appsawg-webfinger
Internet Drafts. It is the intention of this document to provide an
alternative point of view intended to add to the overall discussion
and evolution of a functional, usable, simple discovery mechanism.

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). Note that other groups may also distribute
working documents as Internet-Drafts. The list of current Internet-
Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on January 9, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 3
2. Motivating Examples ........................................... 3
3. The WebFinger Approach ...................................... 4
4. Issues with the WebFinger Approach ........................... 7
5. The Simple Web Discovery Approach ........................... 8
6. Issues with the Simple Web Discovery Approach .......... 8
7. The /.well-known/index Approach ............................ 10
8. Recommended Actions ......................................... 11
9. Security Considerations ...................................... 12
10. Normative References ....................................... 12
Author’s Address .................................................. 12
1. Introduction

There are currently two Internet Drafts currently on the table for discussion within the Applications Area Working Group (appsawg) that deal with the basic discovery of information relating to Web hosted resources. These are:

- The Simple Web Discovery (SWD) mechanism described in [I-D.jones-simple-web-discovery], and
- The WebFinger Protocol (WF) described in [I-D.jones-appsawg-webfinger].

While both present fairly reasonable solutions to the general problem of resource discovery, I believe that there are a number of ways in which both models can be improved upon and brought together into a single simple and flexible discovery mechanism. It is the intent of this memo to describe that alternative approach and contrast it against both the Simple Web Discovery and WebFinger approaches.

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

2. Motivating Examples

Prior to examining the technical details of the various discovery mechanisms being considered, it is important to lay out the fundamental problem for which these specifications are seeking to provide a solution. For this purpose, consider the following scenario:

Assume we have a user, "Joe" who just purchased a new mobile computing device -- a tablet, for instance. After excitedly opening the box and powering on the device, he is presented with a welcome screen informing him that the device is capable of accessing and utilizing a variety of online services that Joe may have access to. All Joe needs to do is provide the device with one or more of the personal account identifiers that represent the various online services he utilizes. Examples may include hosted email and calendar services, social networking sites, cloud storage providers, document editing and management services, news and entertainment providers, etc. In order to keep the "unboxing experience" for Joe and his new tablet as simple and enjoyable as possible, it is ideal that the tablet software perform as much of the initial discovery and configuration of services as possible with only a minimal amount of input and action required of Joe.

To accomplish this automatic configuration, Joe is asked to provide
one or more account identifiers, each in a form similar to that of an email address. For example, "joe.smith@online-service-provider.example.org". The device configuration software will take this identifier and convert it into an HTTP request sent to http://online-service-provider.example.org -- the intent of which is to discover any information about user "joe.smith" that the server might be willing to make available, including the type and location of any services provided to the user.

There are several important considerations to keep in mind when the device sends this service discovery request to the service provider:

1. In many cases, there are inherent privacy concerns and legal issues that must be considered by the service provider prior to disclosing any information about the individuals making use of their services. As a general rule, service providers SHOULD only provide information about a user when such disclosure is directly authorized by the user, or when such disclosure does not represent a potential breach of the user's privacy.

2. Requests for information about a user SHOULD be limited to the absolute minimum amount of data necessary to complete the immediately necessary task. Discovered information SHOULD be readily cacheable for reasonably long periods of time and SHOULD NOT impose unnecessary burdens on either the requestor or the service provider. Complexity needs to be kept to an absolute minimum.

3. This sort of discovery is definitely not a new problem. There is a deep well of prior experience we can draw from and learn from to inform our development of a reasonable solution to this basic problem.

3. The WebFinger Approach


Step one is for the user (Joe) to provide the WebFinger client with what is known as an "acct" URI -- essentially an email address in the form "joe.smith@online-service-provider.example.org". Upon receiving the acct ID, the client application extracts the domain name (online-service-provider.example.org) and sends an HTTP request asking for that server’s "host-meta" document as defined in [RFC6415].

GET /.well-known/host-meta HTTP/1.1
Host: example.org

The server then responds by returning either an XML-formatted "XRD" document (as described by
or JSON-formatted "ERD" document (as described in [RFC6415]). Within this document would be a link to a so-called "LRDD" resource that is capable of providing the WebFinger client with information about the user "joe.smith".

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/xrd+xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<XRD xmlns="http://docs.oasis-open.org/ns/xri/xrd-1.0">
  <Link rel="lrdd"
       type="application/xrd+xml"
       template="https://example.org/lrdd/?uri={uri}"/>
</XRD>

At this point, the WebFinger client still does not yet know anything specific about the user "joe.smith". It must first parse the returned XML document, locate the "lrdd" link (if it exists at all in the response), and process the provided URL Template [RFC6570] before it can prepare a second HTTP request to retrieve the information it desires about the user.

After doing all that, the WebFinger client can proceed to ask the server for information about the user.

GET /lrdd/?uri=acct:bob.smith@online-service-provider.example.org
Host: example.org

The response to this document is another XML or JSON document that must then be parsed and processed by the WebFinger client.

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/xrd+xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<XRD xmlns="http://docs.oasis-open.org/ns/xri/xrd-1.0">
  <Expires>2012-03-13T20:56:11Z</Expires>
  <Subject>...</Subject>
  <Link rel="http://webfinger.net/rel/avatar"
        href="http://profile.example.org/joe.smith/picture.jpg"/>
  <Link rel="http://webfinger.net/rel/profile-page"
        href="http://profile.example.org/joe.smith"/>
  <Link rel="http://webfinger.net/rel/calendar"
        href="http://calendar.example.org/joe.smith"/>
</XRD>
At this point, the WebFinger Client has had to send two separate HTTP requests and parse two separate documents in order to learn that there are only three resources relating to the user in question.

Recognizing the inefficiency inherent in this design, the authors of the WebFinger specification included a short-cut mechanism that eliminates one of the HTTP requests.

```
GET /.well-known/host-meta?resource=acct:joe.smith@online-service-provider.example.org HTTP/1.1
Host: example.org
```

When used in this way, the server SHOULD essentially perform the second query operation on the WebFinger client’s behalf, returning the second XRD document above in response rather than the domains generic "host-meta" document.

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/xrd+xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<XRD xmlns="http://docs.oasis-open.org/ns/xri/xrd-1.0">
  <Expires>2012-03-13T20:56:11Z</Expires>
  <Subject>...</Subject>
  <Link rel="http://webfinger.net/rel/avatar" href="http://profile.example.org/joe.smith/picture.jpg"/>
  <Link rel="http://webfinger.net/rel/profile-page" href="http://profile.example.org/joe.smith/"/>
  <Link rel="http://webfinger.net/rel/calendar" href="http://calendar.example.org/joe.smith"/>
</XRD>
```

Another inefficiency inherent in the original WebFinger design is the fact that a requesting client must parse the returned document in order to determine which services are available for the identified user. However, it is often the case that the requesting client knows in advance exactly which service or services it is interested in knowing about. Ideally, the protocol would provide a means of allowing the client to ask specifically for information relating to just that service or set of services, ignoring all others that it may not care about.

To that end, more recent versions of the WebFinger protocol have introduced a second optional "rel" parameter that can be included in the "host-meta" request to specify the specific services the client is interested in learning about.
GET /.well-known/host-meta?resource=acct:joe.smith@online-service-provider.example.org&rel=avatar+calendar HTTP/1.1
Host: online-service-provider.example.org

The server would then be expected to return an XRD or JRD document containing only information about the identified resource and services.

4. Issues with the WebFinger Approach

While the WebFinger Protocol does provide a basically workable solution to the problem and does have working, deployed implementations in the wild, there are a number of glaring issues that, in my opinion, make it less than ideal approach.

First is the dependency on XRD and JRD. The requirement to support two alternative serializations of the same response data unnecessarily adds complexity. The WebFinger Protocol should draw a line in the sand and just pick one <http://www.mnot.net/blog/2012/04/13/json_or_xml_just_decide>.

Second is the unnecessary two HTTP request process to discover service information for a user. The Web Finger specification itself recognizes that the additional indirection caused by the host-meta and LRDD service lookups is both inefficient and unnecessary given the fact that it defines a shortcut around both in the form of the resource and rel query parameters. If the requirement can be addressed by sending a single HTTP request and processing a single response document, then why is there even a need to define the more inefficient two-step mechanism?

Third, the WebFinger protocol defines normative dependencies on no fewer than ten separate specifications. A WebFinger client needs to not only be capable of sending HTTP requests; but capable of parsing XML or JSON; capable of understanding the specific XRD and JRD vocabularies; capable of processing URL Templates; and capable of processing XML digital signatures included within an XRD document (per section 11). This seems entirely too excessive and unnecessary given the basic requirements of the original problem. There is a much simpler way of achieving the goal.

Fourth, WebFinger fails to address one fundamental privacy issue: many organizations have established policies against utilizing clearly-readable email-like identifiers within HTTP request URIs (<http://goo.gl/meSUZ>). Such use presents a potential threat to the privacy of users of the service.
5. The Simple Web Discovery Approach

The Simple Web Discovery [I-D.jones-simple-web-discovery] specification takes an alternative, simpler approach to addressing the same fundamental problem.

Step one, is for the client application to send a single HTTP request to the server specifying the identity of the user in question and the service the client is interested in knowing about.

GET /.well-known/simple-web-discovery
    ?principal=mailto:joe.smith@online-service-provider.example.com
    &service=urn:example:service:calendar HTTP/1.1
Host: example.com

The server would respond with a simple JSON document providing the URL of the requested service.

HTTP/1.1 200 OK
Content-Type: application/json

{
    "locations": ["https://calendars.example.org/joe.smith"]
}

If the server wishes, it can choose to redirect the request to a separate HTTP URL. This is done, not by making use of HTTP’s inherent redirection mechanisms (e.g. a 302 Found response) but by returning a JSON document containing the URL of the alternative endpoint.

HTTP/1.1 200 OK
Content-Type: application/json

{
    "SWD_service_redirect": {
        "location": "https://swd.example.com/swd_server",
        "expires": 1300752001
    }
}

Upon receiving such a response, the client would need to reissue their request to the new location.

6. Issues with the Simple Web Discovery Approach

While the Simple Web Discovery (SWD) specification is significant
less complicated than the WebFinger approach, there are a number of issues of concern.

First, as was the case with WebFinger, SWD does not address the fundamental privacy issue caused by the use of email-like identifiers within the request URI.

Second, SWD allows the client to request information about only one service at a time. This means that, unlike WebFinger, if a client wishes to know about multiple available for a given user, multiple HTTP requests must be sent and that the client must have a reasonable expectation, in advance, about which services are provided by a given server.

Third, JSON-responses provided by the SWD server are a crude imitation of the redirection capabilities that are inherently built in to the HTTP protocol. For instance, the response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "locations": ["https://calendars.example.org/joe.smith"]
}
```

Could easily be recast into the form of a 302 Found Redirect:

```
HTTP/1.1 302 Found
Location: https://calendars.example.org/joe.smith
```

Doing so would eliminate the need for the client to parse the JSON payload and would allow the client to take full advantage of the redirected request support that is naturally built into HTTP.

For example, consider an alternative scenario where the client is requesting a link to the user’s avatar image rather than their calendar. Using an HTTP redirect rather than a JSON-based document in the response would allow the client to do simply:

```
<img src="http://example.org/.well-known/simple-web-discovery\?
principal=mailto:joe.smith@online-service-provider.example.org&
service=urn:example:service:avatar" />
```

Likewise, the SWD service redirection response can be converted into an HTTP redirection response:

```
HTTP/1.1 302 Found
Location: https://swd.example.com/swd_server
```
While SWD is certainly a step in the right direction compared to WebFinger, there are a number of obvious ways in which the protocol can be improved.

7. The /.well-known/index Approach

The "/.well-known/index" is an alternative solution that I propose is a reasonable and simplified evolution of the Simple Web Discovery approach that addresses not only the original problem but the inherent weaknesses within the previously discussed WebFinger and SWD approaches.

Like WebFinger and SWD, index makes use of the .well-known URL capability defined by [RFC5785]. However, rather than using query-string parameters, the index uses hierarchical path segments to identify the information being requested. Additionally, rather than transmitting email-like identifiers in plain text within the request URI, index calls for such identifiers to be hashed.

For instance, given that the SHA-256 digest of the value "acct:joe.smith@online-service-provider.example.org" is "53ae56ef33cccb9550869e58820df36c3b1cc9574712556059a3bfc716b4d9255", a request to retrieve a redirection to the calendar service for user "joe.smith" would be:

GET /.well-known/index/
  53ae56ef33cccb9550869e58820df36c3b1cc9574712556059a3bfc716b4d9255/
  calendar
Host: online-service-provider.example.org

The response to this request would be an HTTP redirect:

HTTP/1.1 302 Found
Location: https://calendar.example.org/joe.smith

If the client wishes to request information about multiple services, each can be specified within the request URL using a comma-separated list. For instance:

GET /.well-known/index/
  53ae56ef33cccb9550869e58820df36c3b1cc9574712556059a3bfc716b4d9255/
  calendar,avatar
Host: online-service-provider.example.org

The response would again be an HTTP redirect using the 300 Multiple
Choices status code and containing a single Link header for each requested service:

HTTP/1.1 300 Multiple Choices
Link: <https://calendar.example.org/joe.smith>; rel="calendar",
     <https://profile.example.org/joe.smith/avatar.jpg>; rel="avatar"

If the client is not aware ahead of time as to what services are available for a given user, a GET request can be issued that omits the service name from the request URI:

GET /.well-known/index/\53ae56ef33cc9550869e58820df36c3b1cc9574712556059a3bfc716b4d9255
Host: online-service-provider.example.org

The response would be a 300 Multiple Choices redirect with Link headers pointing to each available service:

HTTP/1.1 300 Multiple Choices
Link: <./calendar>; rel="calendar",
     <./avatar>; rel="avatar",
     <./profile>; rel="profile"

With this approach, it remains entirely possible for a server to serve up additional detail in the form of a document describing the service (e.g. an XRD), if it so desires, at the target end of the HTTP redirect.

It also remains possible, through appropriate use of HTTP Caching mechanisms, for redirect responses to be cached for reasonably long periods of time.

This approach introduces no extraneous normative dependencies on multiple external specifications such as XML, JSON, XRD, Host-meta, URL Templates, etc. At a minimum, a client application needs only to know to send HTTP requests, process Link headers and generate SHA-256 digests of account identifiers in order to function. Additional mechanisms, such as OAuth 2.0, can easily be applied as necessary.

8. Recommended Actions

Of the two draft specifications on the table currently, the Simple Web Discovery specification is by far the better choice. The WebFinger specification should be tabled for now in favor of an evolved version of Simple Web Discovery that adopts (either wholly or in part) the index solution described herein.
Specifically, at a minimum, the Simple Web Discovery specification should be updated such that:

1. Account identifiers can be specified in the form of digest values rather than the literal, plain-text identifiers, and
2. Multiple services can be specified, or
3. No services can be specified in the case the client wishes to discover all available information about a given identifier, and
4. HTTP-level redirects are used rather than the JSON-based "redirection" documents.

9. Security Considerations

None.

10. Normative References

[I-D.jones-appsawg-webfinger]
Jones, P., Salgueiro, C., and J. Smarr, "WebFinger",
draft-jones-appsawg-webfinger-06 (work in progress),
June 2012.

[I-D.jones-simple-web-discovery]
Jones, M. and Y. Goland, "Simple Web Discovery (SWD)",
draft-jones-simple-web-discovery-03 (work in progress),
July 2012.


Author’s Address

James M Snell

Email: jasnell@gmail.com