

appsawg
Internet-Draft
Intended status: Experimental
Expires: December 20, 2014

L. Goix
Telecom Italia
K. Li
Huawei Technologies
June 18, 2014

ENUM Service Registration for acct URI
draft-goix-appsawg-enum-acct-uri-07

Abstract

This document registers a Telephone Number Mapping (ENUM) service for 'acct:' URIs (Uniform Resource Identifiers).

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on December 20, 2014.

Copyright Notice

Copyright (c) 2014 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	2
2. Terminology	2
3. Use cases	2
3.1. Reverse phone lookup	2
3.2. Routing of mobile social communications	3
4. IANA Registration	3
5. Examples	4
6. DNS Considerations	5
7. Security Considerations	5
8. IANA Considerations	6
9. Acknowledgements	7
10. References	7
10.1. Normative References	7
10.2. Informative References	8
Authors' Addresses	8

1. Introduction

ENUM (E.164 Number Mapping, [RFC6116]) is a system that uses DNS (Domain Name Service, [RFC1034]) to translate telephone numbers, such as '+44 1632 960123', into URIs (Uniform Resource Identifiers, [RFC3986]), such as 'acct:user@example.com'. ENUM exists primarily to facilitate the interconnection of systems that rely on telephone numbers with those that use URIs to identify resources.

[I-D.ietf-appsawg-acct-uri] defines the 'acct' URI scheme as a way to identify a user's account at a service provider.

This document registers an Enumservice for advertising acct URI information associated with an E.164 number.

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 [RFC2119].

3. Use cases

3.1. Reverse phone lookup

In this example, an address book application could issue ENUM queries looking for 'acct' URIs corresponding to phone numbers. This could be used to display the account identifier as well as an icon based on the host (domain) portion of that URI.

Similarly, an endpoint could trigger this resolution process during inbound and/or outbound calls to discover an account associated with the remote party.

In general the provision of an ENUM record to map a phone number into an account may be useful for businesses or professional workers to identify themselves publicly (in a similar way as vCard enum records).

3.2. Routing of mobile social communications

The Open Mobile Alliance (OMA) develops mobile service enabler specifications, which support the creation of interoperable end-to-end mobile services independent of the underlying wireless platforms, such as GSM (Global System for Mobile communications), UMTS (Universal Mobile Telecommunications System) and LTE (Long Term Evolution) mobile networks. The OMA Social Network Web (SNeW) Enabler Release [OMA-SNeW] has introduced a number of Social Networking functionalities for mobile subscribers identified by their MSISDN (Mobile Subscriber Integrated Services Digital Network number, a number uniquely identifying a subscription in a mobile network), amongst which is the ability to follow each other's social activities across service providers.

Such functionality requires the global resolution of the MSISDN to the corresponding account and provider, in an analogous way as MMS routing, to identify the target endpoint for the related messages. Although alternatives solutions exist (e.g. based on mobile network operations and/or proprietary lookup techniques), ENUM provides a globally accessible mechanism for enabling resolution from network entities on behalf of an endpoint, or from an endpoint itself.

For example, a user of a service provider could request to follow the social activities of user '+44 1632 960123'. The home SNEW Server of the former user could perform an ENUM query to identify the 'acct' URI corresponding to that phone number. Based on the resulting URI, the server could then identify the SNEW Server of the target user and route the original user's request to the appropriate endpoint.

A similar mechanism can apply to other types of social networking-related messages or other communications targeted to a mobile subscriber.

4. IANA Registration

As defined in [RFC6117], the following is a template covering information needed for the registration of the Enumservice specified in this document:

```
<record>
  <class>Application-Based, Ancillary</class>
  <type>acct</type>
  <urischeme>acct</urischeme>
  <functionalspec>
    <paragraph>
      This Enumservice indicates that the resource
      can be identified by the associated 'acct' URI
<xref target='I-D.ietf-appsawg-acct-uri' />.
    </paragraph>
  </functionalspec>
  <security>
    For DNS considerations in avoiding loops when
    searching for "acct" NAPTRs,
    see <xref type="rfc" data="rfcTHIS"/>,
    <xref target="dns">Section 6</xref>.
    For security considerations,
    see <xref type="rfc" data="rfcTHIS"/>,
    <xref target="security">Section 7</xref>.
  </security>
  <usage>COMMON</usage>
  <registrationdocs>
    <xref type="rfc" data="rfcTHIS"/>
  </registrationdocs>
  <requesters>
    <xref type="person" data="Laurent_Walter_Goix"/>
  </requesters>
</record>

<people>
  <person id="Laurent_Walter_Goix">
    <name>Laurent-Walter Goix</name>
    <org>Telecom Italia</org>
    <uri>mailto:laurentwalter.goix@telecomitalia.it</uri>
    <updated>2014-06-18</updated>
  </person>
</people>
```

[Note for RFC-Editor: Please replace any instance of rfcTHIS with the RFC number of this document before publication]

5. Examples

The following is an example of the use of the Enumservice registered by this document in a NAPTR resource record for phone number +44 1632 960123.

\$ORIGIN 3.2.1.0.6.9.2.3.6.1.4.4.e164.arpa.

```
IN NAPTR 10 100 "u" "E2U+acct" "!^.*$!acct:441632960123@foo.com!" .
```

```
IN NAPTR 10 101 "u" "E2U+acct" "!^.*$!acct:john.doe@example.com!" .
```

Note that in the first record, the revealed information is limited to the domain of the service provider serving that user as the userpart of the acct URI simply replicates the phone number.

6. DNS Considerations

There may not be any "E2U+acct" NAPTRs returned in response to the original ENUM query on the requested telephone number, but other terminal ENUM NAPTRs that include tel: URLs [RFC3966] (e.g., "voice:tel" or "pstn:tel" or "SMS:tel" or "MMS:tel" - see [RFC6118]) may be present.

The application that made that ENUM query may choose to re-submit ENUM queries for any E.164 numbers included in those returned terminal NAPTRs. Doing so may cause a query loop (e.g., the ENUM records returned from subsequent queries may refer to the telephone number already considered). If applications choose to perform subsequent ENUM queries using telephone numbers retrieved from earlier queries, these applications MUST be aware of the potential for query loops, and MUST be prepared to abort the set of queries if such a loop is detected.

This is a similar issue to the referential loop issue caused by processing non-terminal NAPTR queries, as mentioned in section 5.2.1 of [RFC6116], and a similar technique to mitigate this issue can be used; an application searching for records with "acct" Enumservice may consider that submitting a chain of more than 5 ENUM queries without finding such a record indicates that a referential loop has been entered, and the chain of queries SHOULD be abandoned.

7. Security Considerations

DNS, as used by ENUM, is a global, distributed database. Should implementers of this specification use e164.arpa or any other publicly available domain as the tree for maintaining PSTN Enumservice data, this information would be visible to anyone anonymously.

Carriers, service providers, and other users may choose not to publish such information in the public e164.arpa tree. They may instead simply publish this in an internal ENUM infrastructure that is only able to be queried by trusted elements of their network, thus limiting threats.

For security considerations that apply to all Enumservices, please refer to [RFC6116], section 7.

It is important to note that the ENUM record itself does not need to contain any personal information but only contains a pointer to an account identifier. This identifier may be queried to discover pointers to personal information (e.g. social network information) endpoints and an authorisation mechanism may be in place in that context with any level of granularity although it is out of scope of this document.

Technically, ENUM records themselves could contain pointers to the same endpoints. However the visibility of ENUM records cannot be controlled based on the requesting entity. In that context the simple mapping of the phone number to the account identifier, notwithstanding the disclosure of the association itself, still enables the reuse of more advanced access policies.

Revealing an 'acct' URI by itself is unlikely to introduce many privacy concerns, although, depending on the structure of the URI, it might reveal the full name or employer of the target. The use of anonymous URIs mitigates this risk.

Unlike a traditional telephone number, the endpoint identified by an 'acct' URI may require that requesting entities provide cryptographic credentials for authentication and authorization before messages are exchanged. ENUM can actually provide far greater protection from unwanted requesting entities than does the existing PSTN, despite the public availability of ENUM records.

More serious security concerns are associated with potential attacks against an underlying system (for example, social network system) using the 'acct' URI. For this reason, underlying system should have a number of security requirements that call for authentication, integrity and confidentiality properties, and similar measures to prevent such attacks. And this is out of scope of this document.

8. IANA Considerations

This document requests the IANA registration of the Enumservice with Type "acct" according to the definitions in this document, [RFC6116] and [RFC6117].

Details of the registration are given in Section 4.

9. Acknowledgements

The authors would like to thank Gonzalo Salgueiro, Paul Jones, Lawrence Conroy, Enrico Marocco, Bert Greevenbosch and Bernie Hoeneisen for their valuable feedback to improve this document.

10. References

10.1. Normative References

- [I-D.ietf-appsawg-acct-uri]
Saint-Andre, P., "The 'acct' URI Scheme", draft-ietf-appsawg-acct-uri-07 (work in progress), January 2014.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.
- [RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC6116] Bradner, S., Conroy, L., and K. Fujiwara, "The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)", RFC 6116, March 2011.
- [RFC6117] Hoeneisen, B., Mayrhofer, A., and J. Livingood, "IANA Registration of Enumservices: Guide, Template, and IANA Considerations", RFC 6117, March 2011.
- [RFC6118] Hoeneisen, B. and A. Mayrhofer, "Update of Legacy IANA Registrations of Enumservices", RFC 6118, March 2011.

10.2. Informative References

[OMA-SNeW]

Open Mobile Alliance, "Social Network Web Enabler", OMA-ER-SNeW-V1_0
http://technical.openmobilealliance.org/Technical/release_program/snew_v1_0.aspx, Aug 2013.

Authors' Addresses

Laurent-Walter Goix
Telecom Italia
Via Golgi, 42
Milano 20133
Italy

Email: laurentwalter.goix@telecomitalia.it

Kepeng Li
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen 518129
P. R. China

Phone: +86-755-28971807
Email: likepeng@huawei.com

6MAN
Internet-Draft
Updates: 3986 (if approved)
Intended status: Standards Track
Expires: June 10, 2013

B. Carpenter
Univ. of Auckland
S. Cheshire
Apple Inc.
R. Hinden
Check Point
December 7, 2012

Representing IPv6 Zone Identifiers in Address Literals and Uniform
Resource Identifiers
draft-ietf-6man-uri-zoneid-06

Abstract

This document describes how the Zone Identifier of an IPv6 scoped address, as defined in RFC 4007, can be represented in a literal IPv6 address and in a Uniform Resource Identifier that includes such a literal address. It updates RFC 3986 accordingly.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on June 10, 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. Specification	4
3. Web Browsers	5
4. Security Considerations	6
5. IANA Considerations	6
6. Acknowledgements	6
7. Change log [RFC Editor: Please remove]	7
8. References	7
8.1. Normative References	7
8.2. Informative References	8
Appendix A. Options Considered	8
Authors' Addresses	10

1. Introduction

The Uniform Resource Identifier (URI) syntax [RFC3986] defined how a literal IPv6 address can be represented in the "host" part of a URI. A subsequent specification [RFC4007] extended the text representation of limited-scope IPv6 addresses such that a zone identifier may be concatenated to a literal address, for purposes described in that RFC. Zone identifiers are especially useful in contexts where literal addresses are typically used, for example during fault diagnosis, when it may be essential to specify which interface is used for sending to a link local address. It should be noted that zone identifiers have purely local meaning within the node where they are defined, often being the same as IPv6 interface names. They are completely meaningless for any other node. Today, they are only meaningful when attached to addresses with less than global scope, but it is possible that other uses might be defined in the future.

RFC 4007 does not specify how zone identifiers are to be represented in URIs. Practical experience has shown that this feature is useful, in particular when using a web browser for debugging with link local addresses, but as it is undefined, it is not implemented consistently in URI parsers or in browsers.

Some versions of some browsers accept the RFC 4007 syntax for scoped IPv6 addresses embedded in URIs, i.e., they have been coded to interpret the "%" sign according to RFC 4007 instead of RFC 3986. Clearly this approach is very convenient for users, although it formally breaches the syntax rules of RFC 3986. The present document defines an alternative approach that respects and extends the rules of URI syntax, and IPv6 literals in general, to be consistent.

Thus, this document updates [RFC3986] by adding syntax to allow a zone identifier to be included in a literal IPv6 address within a URI.

It should be noted that in other contexts than a user interface, a zone identifier is mapped into a numeric zone index or interface number. The MIB textual convention InetZoneIndex [RFC4001] and the socket interface [RFC3493] define this as a 32 bit unsigned integer. The mapping between the human-readable zone identifier string and the numeric value is a host-specific function that varies between operating systems. The present document is concerned only with the human-readable string.

Several alternative solutions were considered while this document was developed. The Appendix briefly describes the various options and their advantages and disadvantages.

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].

2. Specification

According to RFC 4007, a zone identifier is attached to the textual representation of an IPv6 address by concatenating "%" followed by <zone_id>, where <zone_id> is a string identifying the zone of the address. However, RFC 4007 gives no precise definition of the character set allowed in <zone_id>. There are no rules or de facto standards for this. For example, the first Ethernet interface in a host might be called %0, %1, %en1, %eth0, or whatever the implementer happened to choose.

In a URI, a literal IPv6 address is always embedded between "[" and "]". This document specifies how a <zone_id> can be appended to the address. Unfortunately "%" is always treated as an escape character in a URI, and according to RFC 3986 it MUST therefore itself be percent-encoded in a URI, in the form "%25". Thus, the scoped address fe80::a%en1 would appear in a URI as http://[fe80::a%25en1].

A <zone_id> SHOULD contain only ASCII characters classified in RFC 3986 as "unreserved". This excludes characters such as "]" or even "%" which would complicate parsing. However, the syntax below does allow such characters to be percent-encoded, for compatibility with existing devices that use them.

If an operating system uses any other characters in zone or interface identifiers that are not in the "unreserved" character set, they MUST be escaped with a "%" sign according to RFC 3986.

We now present the necessary formal syntax.

In RFC 3986, the IPv6 literal format is formally defined in ABNF [RFC5234] by the following rule:

```
IP-literal = "[" ( IPv6address / IPvFuture  ) "]"
```

To provide support for a zone identifier, the existing syntax of IPv6address is retained, and a zone identifier may be added optionally to any literal address. This allows flexibility for unknown future uses. The rule quoted above from RFC 3986 is replaced by three rules:

```
IP-literal = "[" ( IPv6address / IPv6addrz / IPvFuture  ) "]"
```

```
ZoneID = 1*( unreserved / pct-encoded )
```

```
IPv6addrz = IPv6address "%25" ZoneID
```

This syntax fills the gap that is described at the end of Section 11.7 of RFC 4007.

The rules in [RFC5952] SHOULD be applied in producing URIs.

RFC 3986 states that URIs have a global scope, but that in some cases their interpretation depends on the end-user's context. URIs including a ZoneID are to be interpreted only in the context of the host where they originate, since the ZoneID is of local significance only.

RFC 4007 offers guidance on how the ZoneID affects interface/address selection inside the IPv6 stack. Note that the behaviour of an IPv6 stack if passed a non-null zone index for an address other than link-local is undefined.

3. Web Browsers

This section discusses how web browsers might handle this syntax extension. Unfortunately there is no formal distinction between the syntax allowed in a browser's input dialogue box and the syntax allowed in URIs. For this reason, no normative statements are made in this section.

Due to the lack of defined syntax, web browsers have been inconsistent in providing for ZoneIDs. Many have no support, but there are examples of ad hoc support. For example, some versions of Firefox allowed the use of a ZoneID preceded by an unescaped "%" character, but this was removed for consistency with RFC 3986. As another example, some versions of Internet Explorer allow use of a ZoneID preceded by a "%" character escaped as "%25", still beyond the syntax allowed by RFC 3986. This syntax extension is in fact used internally in the Windows operating system and some of its APIs.

It is desirable for all browsers to recognise a ZoneID preceded by an escaped "%". In the spirit of "be liberal with what you accept", we also suggest that URI parsers accept bare "%" signs when possible (i.e., a "%" not followed by two valid and meaningful hexadecimal characters). This would make it possible for a user to copy and paste a string such as "fe80::a%e1" from the output of a "ping" command and have it work. On the other hand, "%e1" would need to be

manually escaped as "fe80::a%25ee1" to avoid any risk of misinterpretation.

Such bare "%" signs are for user interface convenience, and need to be turned into properly escaped characters (where "%25" encodes "%") before the URI is used in any protocol or HTML document. However, URIs including a ZoneID have no meaning outside the originating node. It would therefore be highly desirable for a browser to remove the ZoneID from a URI before including that URI in an HTTP request.

The normal diagnostic usage for the ZoneID syntax will cause it to be entered in the browser's input dialogue box. Thus, URIs including a ZoneID are unlikely to be encountered in HTML documents. However, if they do (for example, in a diagnostic script coded in HTML) it would be appropriate to treat them exactly as above.

4. Security Considerations

The security considerations of [RFC3986] and [RFC4007] apply. In particular, this URI format creates a specific pathway by which a deceitful zone index might be communicated, as mentioned in the final security consideration of RFC 4007. It is emphasised that the format is intended only for debugging purposes, but of course this intention does not prevent misuse.

To limit this risk, implementations MUST NOT allow use of this format except for well-defined usages such as sending to link local addresses under prefix fe80::/10. At the time of writing, this is the only well-defined usage known.

An HTTP client, proxy or other intermediary MUST remove any ZoneID attached to an outgoing URI, as it only has local significance at the sending host.

5. IANA Considerations

This document requests no action by IANA.

6. Acknowledgements

The lack of this format was first pointed out by Margaret Wasserman some years ago, and more recently by Kerry Lynn. A previous draft document by Martin Duerst and Bill Fenner [I-D.fenner-literal-zone] discussed this topic but was not finalised.

Valuable comments and contributions were made by Karl Auer, Carsten Bormann, Benoit Claise, Stephen Farrell, Brian Haberman, Ted Hardie, Tatuya Jinmei, Yves Lafon, Barry Leiba, Radia Perlman, Tom Petch, Tomoyuki Sahara, Juergen Schoenwaelder, Dave Thaler, Martin Thomson, and Ole Troan.

Brian Carpenter was a visitor at the Computer Laboratory, Cambridge University during part of this work.

This document was produced using the xml2rfc tool [RFC2629].

7. Change log [RFC Editor: Please remove]

draft-ietf-6man-uri-zoneid-06: responding to IETF Last Call and IESG comments, 2012-12-07.

draft-ietf-6man-uri-zoneid-05: tuned ABNF, clarified RFC 4007 text, 2012-11-06.

draft-ietf-6man-uri-zoneid-04: additional author, 2012-09-21.

draft-ietf-6man-uri-zoneid-03: reverted to percent-encoded model following WGLC, 2012-09-10.

draft-ietf-6man-uri-zoneid-02: additional WG comments, 2012-07-11.

draft-ietf-6man-uri-zoneid-01: use "-" instead of %25, listed alternatives in Appendix, according to WG debate, added suggestion for browser developers, 2012-05-29.

draft-ietf-6man-uri-zoneid-00: adopted by WG, fixed syntax to allow for % encoded characters, 2012-02-17.

draft-carpenter-6man-uri-zoneid-01: chose Option 2, removed 15 character limit, added explanation of ID/number mapping and other clarifications, 2012-02-08.

draft-carpenter-6man-uri-zoneid-00: original version, 2011-12-07.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, March 2005.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

8.2. Informative References

- [I-D.fenner-literal-zone]
Fenner, B. and M. Duerst, "Formats for IPv6 Scope Zone Identifiers in Literal Address Formats", draft-fenner-literal-zone-02 (work in progress), October 2005.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.
- [RFC4001] Daniele, M., Haberman, B., Routhier, S., and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 4001, February 2005.
- [chrome] Google, "Use the address bar (omnibox)", 2012, <<http://support.google.com/chrome/bin/answer.py?answer=95440>>.

Appendix A. Options Considered

The syntax defined above allows a ZoneID to be added to any IPv6 address. The 6man WG discussed and rejected an alternative in which the existing syntax of IPv6address would be extended by an option to add the ZoneID only for the case of link-local addresses. It was felt that the present solution offers more flexibility for future uses and is more straightforward to implement.

The various syntax options considered are now briefly described.

1. Leave the problem unsolved.

This would mean that per-interface diagnostics would still have to be performed using ping or ping6:

```
ping fe80::a%en1
```

Advantage: works today.

Disadvantage: less convenient than using a browser.

2. Simply using the percent character.

```
http://[fe80::a%en1]
```

Advantage: allows use of browser, allows cut and paste.

Disadvantage: invalid syntax under RFC 3986; not acceptable to URI community.

3. Escaping the escape character as allowed by RFC 3986:

```
http://[fe80::a%25en1]
```

Advantage: allows use of browser, consistent with general URI syntax.

Disadvantage: somewhat ugly and confusing, doesn't allow simple cut and paste.

This is the option chosen for standardization.

4. Alternative separator

```
http://[fe80::a-en1]
```

Advantage: allows use of browser, simple syntax

Disadvantage: Requires all IPv6 address literal parsers and generators to be updated in order to allow simple cut and paste; inconsistent with existing tools and practice.

Note: the initial proposal for this choice was to use an underscore as the separator, but it was noted that this becomes effectively invisible when a user interface automatically underlines URLs.

5. With the "IPvFuture" syntax left open in RFC 3986:

`http://[v6.fe80::a_en1]`

Advantage: allows use of browser.

Disadvantage: ugly and redundant, doesn't allow simple cut and paste.

Authors' Addresses

Brian Carpenter
Department of Computer Science
University of Auckland
PB 92019
Auckland, 1142
New Zealand

Email: brian.e.carpenter@gmail.com

Stuart Cheshire
Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
US

Email: cheshire@apple.com

Robert M. Hinden
Check Point Software Technologies, Inc.
800 Bridge Parkway
Redwood City, CA 94065
US

Email: bob.hinden@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 27, 2014

P. Saint-Andre
January 23, 2014

The 'acct' URI Scheme
draft-ietf-appsawg-acct-uri-07

Abstract

This document defines the 'acct' Uniform Resource Identifier (URI) scheme as a way to identify a user's account at a service provider, irrespective of the particular protocols that can be used to interact with the account.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on July 27, 2014.

Copyright Notice

Copyright (c) 2014 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	2
2. Terminology	2
3. Rationale	2
4. Definition	3
5. Security Considerations	4
6. Internationalization Considerations	5
7. IANA Considerations	5
8. References	7
Appendix A. Acknowledgements	8
Author's Address	8

1. Introduction

Existing Uniform Resource Identifier (URI) schemes that enable interaction with, or that identify resources associated with, a user's account at a service provider are tied to particular services or application protocols. Two examples are the 'mailto' scheme (which enables interaction with a user's email account) and the 'http' scheme (which enables retrieval of web files controlled by a user or interaction with interfaces providing information about a user). However, there exists no URI scheme that generically identifies a user's account at a service provider without specifying a particular protocol to use when interacting with the account. This specification fills that gap.

2. Terminology

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

3. Rationale

During formalization of the WebFinger protocol [RFC7033], much discussion occurred regarding the appropriate URI scheme to include when specifying a user's account as a web link [RFC5988]. Although both the 'mailto' [RFC6068] and 'http' [RFC2616] schemes were proposed, not all service providers offer email services or web interfaces on behalf of user accounts (e.g., a microblogging or instant messaging provider might not offer email services, or an enterprise might not offer HTTP interfaces to information about its employees). Therefore, the participants in the discussion recognized that it would be helpful to define a URI scheme that could be used to generically identify a user's account at a service provider, irrespective of the particular application protocols used to interact

with the account. The result was the 'acct' URI scheme defined in this document.

(Note that a user is not necessarily a human; it could be an automated application such as a bot, a role-based alias, etc. However, an 'acct' URI is always used to identify something that has an account at a service, not the service itself.)

4. Definition

The syntax of the 'acct' URI scheme is defined under Section 7 of this document. Although 'acct' URIs take the form "user@host", the scheme is designed for the purpose of identification instead of interaction (regarding this distinction, see Section 1.2.2 of [RFC3986]). The "Internet resource" identified by an 'acct' URI is a user's account hosted at a service provider, where the service provider is typically associated with a DNS domain name. Thus a particular 'acct' URI is formed by setting the "user" portion to the user's account name at the service provider and by setting the "host" portion to the DNS domain name of the service provider.

Consider the case of a user with an account name of "foobar" on a microblogging service "status.example.net". It is taken as convention that the string "foobar@status.example.net" designates that account. This is expressed as a URI using the 'acct' scheme as "acct:foobar@status.example.net".

A common scenario is for a user to register with a service provider using an identifier (such as an email address) that is associated with some other service provider. For example, a user with the email address "juliet@capulet.example" might register with a commerce website whose domain name is "shoppingsite.example". In order to use her email address as the localpart of the 'acct' URI, the at-sign character (U+0040) needs to be percent-encoded as described in [RFC3986]. Thus the resulting 'acct' URI would be "acct:juliet%40capulet.example@shoppingsite.example".

It is not assumed that an entity will necessarily be able to interact with a user's account using any particular application protocol, such as email; to enable such interaction, an entity would need to use the appropriate URI scheme for such a protocol, such as the 'mailto' scheme. While it might be true that the 'acct' URI minus the scheme name (e.g., "user@example.com" derived from "acct:user@example.com") can be reached via email or some other application protocol, that fact would be purely contingent and dependent upon the deployment practices of the provider.

Because an 'acct' URI enables abstract identification only and not interaction, this specification provides no method for dereferencing an 'acct' URI on its own, e.g., as the value of the 'href' attribute of an HTML anchor element. For example, there is no behavior specified in this document for an 'acct' URI used as follows:

```
<a href='acct:bob@example.com'>find out more</a>
```

Any protocol that uses 'acct' URIs is responsible for specifying how an 'acct' URI is employed in the context of that protocol (in particular, how it is dereferenced or resolved; see [RFC3986]). As a concrete example, an "Account Information" application of the WebFinger protocol [RFC7033] might take an 'acct' URI, resolve the host portion to find a WebFinger server, and then pass the 'acct' URI as a parameter in a WebFinger HTTP request for metadata (i.e., web links [RFC5988]) about the resource. For example:

```
GET /.well-known/webfinger?resource=acct%3Abob%40example.com HTTP/1.1
```

The service retrieves the metadata associated with the account identified by that URI and then provides that metadata to the requesting entity in an HTTP response.

If an application needs to compare two 'acct' URIs (e.g., for purposes of authentication and authorization), it MUST do so using case normalization and percent-encoding normalization as specified in Sections 6.2.2.1 and 6.2.2.2 of [RFC3986].

5. Security Considerations

Because the 'acct' URI scheme does not directly enable interaction with a user's account at a service provider, direct security concerns are minimized.

However, an 'acct' URI does provide proof of existence of the account; this implies that harvesting published 'acct' URIs could prove useful to spammers and similar attackers, for example if they can use an 'acct' URI to leverage more information about the account (e.g., via WebFinger) or if they can interact with protocol-specific URIs (such as 'mailto' URIs) whose user@host portion is the same as that of the 'acct' URI.

In addition, protocols that make use of 'acct' URIs are responsible for defining security considerations related to such usage, e.g., the risks involved in dereferencing an 'acct' URI, the authentication and authorization methods that could be used to control access to

personal data associated with a user's account at a service, and methods for ensuring the confidentiality of such information.

The use of percent-encoding allows a wider range of characters in account names, but introduces some additional risks. Implementers are advised to disallow percent-encoded characters or sequences that would (1) result in space, null, control, or other characters that are otherwise forbidden, (2) allow unauthorized access to private data, or (3) lead to other security vulnerabilities.

6. Internationalization Considerations

As specified in [RFC3986], the 'acct' URI scheme allows any character from the Unicode repertoire [UNICODE] encoded as UTF-8 [RFC3629] and then percent-encoded into valid ASCII [RFC20]. Before applying any percent-encoding, an application MUST ensure the following about the string that is used as input to the URI-construction process:

- o The userpart consists only of Unicode code points that conform to the PRECIS IdentifierClass specified in [I-D.ietf-precis-framework].
- o The host consists only of Unicode code points that conform to the rules specified in [RFC5892].
- o Internationalized domain name (IDN) labels are encoded as A-labels [RFC5890].

7. IANA Considerations

In accordance with the guidelines and registration procedures for new URI schemes [RFC4395], this section provides the information needed to register the 'acct' URI scheme.

7.1. URI Scheme Name

acct

7.2. Status

permanent

7.3. URI Scheme Syntax

The 'acct' URI syntax is defined here in Augmented Backus-Naur Form (ABNF) [RFC5234], borrowing the 'host', 'pct-encoded', 'sub-delims', 'unreserved' rules from [RFC3986]:


```
acctURI      = "acct" ":" userpart "@" host
userpart     = unreserved / sub-delims
              0*( unreserved / pct-encoded / sub-delims )
```

Note that additional rules regarding the strings that are used as input to construction of 'acct' URIs further limit the characters that can be percent-encoded; see the Encoding Considerations as well as Section 6 of RFC XXXX. [Note to RFC Editor: please replace XXXX with the number issued to this document.]

7.4. URI Scheme Semantics

The 'acct' URI scheme identifies accounts hosted at service providers. It is used only for identification, not interaction. A protocol that employs the 'acct' URI scheme is responsible for specifying how an 'acct' URI is dereferenced in the context of that protocol. There is no media type associated with the 'acct' URI scheme.

7.5. Encoding Considerations

See Section 6 of RFC XXXX. [Note to RFC Editor: please replace XXXX with the number issued to this document.]

7.6. Applications/Protocols That Use This URI Scheme Name

At the time of this writing, only the WebFinger protocol uses the 'acct' URI scheme. However, use is not restricted to the WebFinger protocol, and the scheme might be considered for use in other protocols.

7.7. Interoperability Considerations

There are no known interoperability concerns related to use of the 'acct' URI scheme.

7.8. Security Considerations

See Section 5 of RFC XXXX. [Note to RFC Editor: please replace XXXX with the number issued to this document.]

7.9. Contact

Peter Saint-Andre, psaintan@cisco.com

7.10. Author/Change Controller

This scheme is registered under the IETF tree. As such, the IETF maintains change control.

7.11. References

None.

8. References

8.1. Normative References

- [I-D.ietf-precis-framework] Saint-Andre, P. and M. Blanchet, "Precis Framework: Handling Internationalized Strings in Protocols", draft-ietf-precis-framework-13 (work in progress), December 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.
- [RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.
- [UNICODE] The Unicode Consortium, "The Unicode Standard, Version 6.1", 2012, <<http://www.unicode.org/versions/Unicode6.1.0/>>.

8.2. Informative References

- [RFC20] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, October 2010.
- [RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, September 2013.

Appendix A. Acknowledgements

The 'acct' URI scheme was originally proposed during work on the WebFinger protocol; special thanks are due to Blaine Cook, Brad Fitzpatrick, and Eran Hammer-Lahav for their early work on the concept (which in turn was partially inspired by work on Extensible Resource Identifiers at OASIS). The scheme was first formally specified in [RFC7033]; the authors of that specification (Paul Jones, Gonzalo Salgueiro, and Joseph Smarr) are gratefully acknowledged. Thanks are also due to Stephane Bortzmeyer, Melvin Carvalho, Martin Duerst, Graham Klyne, Barry Leiba, Subramanian Moonesamy, Evan Prodromou, James Snell, and various participants in the IETF APPSAWG for their feedback. Meral Shirazipour completed a Gen-ART review. Dave Cridland completed an AppsDir review, and is gratefully acknowledged for providing proposed text that was incorporated into Section 3 and Section 5. IESG comments from Richard Barnes, Adrian Farrel, Stephen Farrell, Barry Leiba, Pete Resnick, and Sean Turner also led to improvements in the specification.

Author's Address

Peter Saint-Andre

Email: ietf@stpeter.im

APPSAWG
Internet-Draft
Intended status: Informational
Expires: May 26, 2014

M. Kucherawy
G. Shapiro
N. Freed
November 22, 2013

Advice for Safe Handling of Malformed Messages
draft-ietf-appsawg-malformed-mail-11

Abstract

Although Internet mail formats have been precisely defined since the 1970s, authoring and handling software often show only mild conformance to the specifications. The malformed messages that result are non-standard. Nonetheless, decades of experience has shown that handling with some tolerance the malformations that result is often an acceptable approach, and is better than rejecting the messages outright as nonconformant. This document includes a collection of the best advice available regarding a variety of common malformed mail situations, to be used as implementation guidance.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on May 26, 2014.

Copyright Notice

Copyright (c) 2013 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
1.1. The Purpose Of This Work	3
1.2. Not The Purpose Of This Work	4
1.3. General Considerations	4
2. Document Conventions	5
2.1. Examples	5
3. Background	5
4. Invariant Content	5
5. Mail Submission Agents	6
6. Line Termination	7
7. Header Anomalies	7
7.1. Converting Obsolete and Invalid Syntaxes	7
7.1.1. Host-Address Syntax	8
7.1.2. Excessive Angle Brackets	8
7.1.3. Unbalanced Angle Brackets	8
7.1.4. Unbalanced Parentheses	8
7.1.5. Commas in Address Lists	9
7.1.6. Unbalanced Quotes	9
7.1.7. Naked Local-Parts	10
7.2. Non-Header Lines	10
7.3. Unusual Spacing	11
7.4. Header Malformations	12
7.5. Header Field Counts	12
7.5.1. Repeated Header Fields	14
7.5.2. Missing Header Fields	15
7.5.3. Return-Path	16
7.6. Missing or Incorrect Charset Information	16
7.7. Eight-Bit Data	17
8. MIME Anomalies	18
8.1. Missing MIME-Version Field	18
8.2. Faulty Encodings	18
9. Body Anomalies	19
9.1. Oversized Lines	19
10. Security Considerations	19
11. IANA Considerations	19
12. References	20
12.1. Normative References	20
12.2. Informative References	20
Appendix A. RFC Editor Notes	21
Appendix B. Acknowledgements	21

1. Introduction

1.1. The Purpose Of This Work

The history of email standards, going back to [RFC733] and beyond, contains a fairly rigid evolution of specifications. However, implementations within that culture have also long had an undercurrent known formally as the robustness principle, also known informally as Postel's Law: "Be liberal in what you accept, and conservative in what you send." [RFC1122]

Jon Postel's directive is often misinterpreted to mean that any deviance from a specification is acceptable. Rather, it was intended only to account for legitimate variations in interpretation within specifications, as well as basic transit errors, like bit errors. Taken to its unintended extreme, excessive tolerance would imply that there are no limits to the liberties that a sender might take, while presuming a burden on a receiver to guess "correctly" at the meaning of any such variation. These matters are further compounded by receiver software -- the end users' mail readers -- which are also sometimes flawed, leaving senders to craft messages (sometimes bending the rules) to overcome those flaws.

In general, this served the email ecosystem well by allowing a few errors in implementations without obstructing participation in the game. The proverbial bar was set low. However, as we have evolved into the current era, some of these lenient stances have begun to expose opportunities that can be exploited by malefactors. Various email-based applications rely on strong application of these standards for simple security checks, while the very basic building blocks of that infrastructure, intending to be robust, fail utterly to assert those standards.

The distributed and non-interactive nature of email has often prompted adjustments to receiving software, to handle these variations, rather than trying to gain better conformance by senders, since the receiving operator is primarily driven by complaints from recipient users and has no authority over the sending side of the system. Processing with such flexibility comes at some cost, since mail software is faced with decisions about whether to permit non-conforming messages to continue toward their destinations unaltered, adjust them to conform (possibly at the cost of losing some of the original message), or outright rejecting them.

This document includes a collection of the best advice available regarding a variety of common malformed mail situations, to be used as implementation guidance. These malformations are typically based around loose interpretations or implementations of specifications

such as Internet Message Format [MAIL] and Multipurpose Internet Mail Extensions [MIME].

1.2. Not The Purpose Of This Work

It is important to understand that this work is not an effort to endorse or standardize certain common malformations. The code and culture that introduces such messages into the mail stream needs to be repaired, as the security penalty now being paid for this lax processing arguably outweighs the reduction in support costs to end users who are not expected to understand the standards. However, the reality is that this will not be fixed quickly.

Given this, it is beneficial to provide implementers with guidance about the safest or most effective way to handle malformed messages when they arrive, taking into consideration the tradeoffs of the choices available especially with respect to how various actors in the email ecosystem respond to such messages in terms of handling, parsing, or rendering to end users.

1.3. General Considerations

Many deviations from message format standards are considered by some receivers to be strong indications that the message is undesirable, such as spam or something containing malware. These receivers quickly decide that the best handling choice is simply to reject or discard the message. This means malformations caused by innocent misunderstandings or ignorance of proper syntax can cause messages with no ill intent also to fail to be delivered.

Senders that want to ensure message delivery are best advised to adhere strictly to the relevant standards (including, but not limited to, [MAIL], [MIME], and [DKIM]), as well as observe other industry best practices such as may be published from time to time either by the IETF or independently.

Receivers that haven't the luxury of strict enforcement of the standards on inbound messages are usually best served by observing the following guidelines for handling of malformed messages:

1. Whenever possible, mitigation of syntactic malformations should be guided by an assessment of the most likely semantic intent. For example, it is reasonable to conclude that multiple sets of angle brackets around an address are simply superfluous and can be dropped.
2. When the intent is unclear, or when it is clear but also impractical to change the content to reflect that intent,

mitigation should be limited to cases where not taking any corrective action would clearly lead to a worse outcome.

3. Security issues, when present, need to be addressed and may force mitigation strategies that are otherwise suboptimal.

2. Document Conventions

2.1. Examples

Examples of message content include a number within braces at the end of each line. These are line numbers for use in subsequent discussion, and are not actually part of the message content presented in the example.

Blank lines are not numbered in the examples.

3. Background

The reader would benefit from reading [EMAIL-ARCH] for some general background about the overall email architecture. Of particular interest is the Internet Message Format, detailed in [MAIL]. Throughout this document, the use of the term "message" should be assumed to mean a block of text conforming to the Internet Message Format.

4. Invariant Content

An agent handling a message could use several distinct representations of the message. One is an internal representation, such as separate blocks of storage for the header and body, some header or body alterations, or tables indexed by header name, set up to make particular kinds of processing easier. The other is the representation passed along to the next agent in the handling chain. This might be identical to the message input to the module, or it might have some changes such as added or reordered header fields or body elisions to remove malicious content.

Message handling is usually most effective when each in a sequence of handling modules receives the same content for analysis. A module that "fixes" or otherwise alters the content passed to later modules can prevent the later modules from identifying malicious or other content that exposes the end user to harm. It is important that all processing modules can make consistent assertions about the content. Modules that operate sequentially sometimes add private header fields to relay information downstream for later filters to use (and possibly remove), or they may have out-of-band ways of doing so. However, even the presence of private header fields can impact a

downstream handling agent unaware of its local semantics, so an out-of-band method is always preferable.

The above is less of a concern when multiple analysis modules are operated in parallel, independent of one another.

Often, abuse reporting systems can act effectively only when a complaint or report contains the original message exactly as it was generated. Messages that have been altered by handling modules might render a complaint inactionable as the system receiving the report may be unable to identify the original message as one of its own.

Some message changes alter syntax without changing semantics. For example, Section 7.4 describes a situation where an agent removes additional header whitespace. This is a syntax change without a change in semantics, though some systems (such as DKIM) are sensitive to such changes. Message system developers need to be aware of the downstream impact of making either kind of change.

Where a change to content between modules is unavoidable, adding trace data (such as prepending a standard Received field) will at least allow tracing of the handling by modules that actually see different input.

There will always be local handling exceptions, but these guidelines should be useful for developing integrated message processing environments.

In most cases, this document only discusses techniques used on internal representations. It is occasionally necessary to make changes between the input and output versions; such cases will be called out explicitly.

5. Mail Submission Agents

Within the email context, the single most influential component that can reduce the presence of malformed items in the email system is the Mail Handling Service (MHS; see [EMAIL-ARCH]), which includes the Mail Submission Agent (MSA). This is the component that is essentially the interface between end users that create content and the mail stream.

MHSes need to become more strict about enforcement of all relevant email standards, especially [MAIL] and the [MIME] family of documents.

More strict conformance by relaying Mail Transfer Agents (MTAs) will also be helpful. although preventing the dissemination of malformed

messages is desirable, the rejection of such mail already in transit also has a support cost, namely the creation of a [DSN] that many end users might not understand.

6. Line Termination

For interoperable Internet Mail messages, the only valid line separation sequence during a typical SMTP session is ASCII 0x0D ("carriage return", or CR) followed by ASCII 0x0A ("line feed", or LF), commonly referred to as CRLF. This is not the case for binary mode SMTP (see [BINARYSMTP]).

Common UNIX user tools, however, typically only use LF for internal line termination. This means that a protocol engine that converts between UNIX and Internet Mail formats has to convert between these two end-of-line representations before transmitting a message or after receiving it.

Non-compliant implementations can create messages with a mix of line terminations, such as LF everywhere except CRLF only at the end of the message. According to [SMTP] and [MAIL], this means the entire message actually exists on a single line.

Within modern Internet Mail it is highly unlikely that an isolated CR or LF is valid in common ASCII text. Furthermore, when content actually does need to contain such an unusual character sequence, [MIME] provides mechanisms for encoding that content in an SMTP-safe manner.

Thus, it will typically be safe and helpful to treat an isolated CR or LF as equivalent to a CRLF when parsing a message.

Note that this advice pertains only to the raw SMTP data, and not to decoded MIME entities. As noted above, when MIME encoding mechanisms are used, the unusual character sequences are not visible in the raw SMTP stream.

7. Header Anomalies

This section covers common syntactic and semantic anomalies found in a message header, and presents suggested mitigations.

7.1. Converting Obsolete and Invalid Syntaxes

A message using an obsolete header syntax (see Section 4 of [MAIL]) might confound an agent that is attempting to be robust in its handling of syntax variations. A bad actor could exploit such a weakness in order to get abusive or malicious content through a

filter. This section presents some examples of such variations. Messages including them ought be rejected; where this is not possible, recommended internal interpretations are provided.

7.1.1. Host-Address Syntax

The following obsolete syntax attempts to specify source routing:

To: <@example.net:fran@example.com>

This means "send to fran@example.com via the mail service at example.net". It can safely be interpreted as:

To: <fran@example.com>

7.1.2. Excessive Angle Brackets

The following over-use of angle brackets:

To: <<<user2@example.org>>>

can safely be interpreted as:

To: <user2@example.org>

7.1.3. Unbalanced Angle Brackets

The following use of unbalanced angle brackets:

To: <another@example.net

can usually be treated as:

To: <another@example.net>

The following:

To: second@example.org>

can usually be treated as:

To: second@example.org

7.1.4. Unbalanced Parentheses

The following use of unbalanced parentheses:

To: (Testing <fran@example.com>

can safely be interpreted as:

To: (Testing) <fran@example.com>

Likewise, this case:

To: Testing) <sam@example.com>

can safely be interpreted as:

To: "Testing)" <sam@example.com>

In both cases, it is obvious where the active email address in the string can be found. The former case retains the active email address in the string by completing what appears to be intended as a comment; the intent in the latter case is less obvious, so the leading string is interpreted as a display name.

7.1.5. Commas in Address Lists

This use of an errant comma:

To: <third@example.net, fourth@example.net>

can usually be interpreted as ending an address, so the above is usually best interpreted as:

To: third@example.net, fourth@example.net

7.1.6. Unbalanced Quotes

The following use of unbalanced quotation marks:

To: "Joe <joe@example.com>

leaves software with no obvious "good" interpretation. If it is essential to extract an address from the above, one possible interpretation is:

To: "Joe <joe@example.com>"@example.net

where "example.net" is the domain name or host name of the handling agent making the interpretation. Another possible interpretation, much simpler and likely more correct, is simply:

To: "Joe" <joe@example.com>

7.1.7. Naked Local-Parts

[MAIL] defines a local-part as the user portion of an email address, and the display-name as the "user-friendly" label that accompanies the address specification.

Some broken submission agents might introduce messages with only a local-part or only a display-name and no properly formed address. For example:

```
To: Joe
```

A submission agent ought to reject this or, at a minimum, append "@" followed by its own host name or some other valid name likely to enable a reply to be delivered to the correct mailbox. Where this is not done, an agent receiving such a message will probably be successful by synthesizing a valid header field for evaluation using the techniques described in Section 7.5.2.

7.2. Non-Header Lines

Some messages contain a line of text in the header that is not a valid message header field of any kind. For example:

```
From: user@example.com {1}
To: userpal@example.net {2}
Subject: This is your reminder {3}
about the football game tonight {4}
Date: Wed, 20 Oct 2010 20:53:35 -0400 {5}
```

```
Don't forget to meet us for the tailgate party! {7}
```

The cause of this is typically a bug in a message generator of some kind. Line {4} was intended to be a continuation of line {3}; it should have been indented by whitespace as set out in Section 2.2.3 of [MAIL].

This anomaly has varying impacts on processing software, depending on the implementation:

1. some agents choose to separate the header of the message from the body only at the first empty line (that is, a CRLF immediately followed by another CRLF);
2. some agents assume this anomaly should be interpreted to mean the body starts at line {4}, as the end of the header is assumed by encountering something that is not a valid header field or folded portion thereof;

3. some agents assume this should be interpreted as an intended header folding as described above and thus simply append a single space character (ASCII 0x20) and the content of line {4} to that of line {3};
4. some agents reject this outright as line {4} is neither a valid header field nor a folded continuation of a header field prior to an empty line.

This can be exploited if it is known that one message handling agent will take one action while the next agent in the handling chain will take another. Consider, for example, a message filter that searches message headers for properties indicative of abusive or malicious content that is attached to a Mail Transfer Agent (MTA) implementing option 2 above. An attacker could craft a message that includes this malformation at a position above the property of interest, knowing the MTA will not consider that content part of the header, and thus the MTA will not feed it to the filter, thus avoiding detection. Meanwhile, the Mail User Agent (MUA) which presents the content to an end user, implements option 1 or 3, which has some undesirable effect.

It should be noted that a few implementations choose option 4 above since any reputable message generation program will get header folding right, and thus anything so blatant as this malformation is likely an error caused by a malefactor.

The preferred implementation if option 4 above is not employed is to apply the following heuristic when this malformation is detected:

1. Search forward for an empty line. If one is found, then apply option 3 above to the anomalous line, and continue.
2. Search forward for another line that appears to be a new header field (a name followed by a colon). If one is found, then apply option 3 above to the anomalous line, and continue.

7.3. Unusual Spacing

The following message is valid per [MAIL]:

```
From: user@example.com {1}
To: userpal@example.net {2}
Subject: This is your reminder {3}
{4}
about the football game tonight {5}
Date: Wed, 20 Oct 2010 20:53:35 -0400 {6}
```

Don't forget to meet us for the tailgate party! {8}

Line {4} contains a single whitespace. The intended result is that lines {3}, {4}, and {5} comprise a single continued header field. However, some agents are aggressive at stripping trailing whitespace, which will cause line {4} to be treated as an empty line, and thus the separator line between header and body. This can affect header-specific processing algorithms as described in the previous section.

This example was legal in earlier versions of the Internet Mail format standard, but was rendered obsolete as of [RFC2822] as line {4} could be interpreted as the separator between the header and body.

The best handling of this example is for a message parsing engine to behave as if line {4} was not present in the message and for a message creation engine to emit the message with line {4} removed.

7.4. Header Malformations

Among the many possible malformations, a common one is insertion of whitespace at unusual locations, such as:

```
From: user@example.com {1}  
To: userpal@example.net {2}  
Subject: This is your reminder {3}  
MIME-Version : 1.0 {4}  
Content-Type: text/plain {5}  
Date: Wed, 20 Oct 2010 20:53:35 -0400 {6}
```

Don't forget to meet us for the tailgate party! {8}

Note the addition of whitespace in line {4} after the header field name but before the colon that separates the name from the value.

The obsolete grammar of Section 4 of [MAIL] permits that extra whitespace, so it cannot be considered invalid. However, a consensus of implementations prefers to remove that whitespace. There is no perceived change to the semantics of the header field being altered as the whitespace is itself semantically meaningless. Therefore, it is best to remove all whitespace after the field name but before the colon and to emit the field in this modified form.

7.5. Header Field Counts

Section 3.6 of [MAIL] prescribes specific header field counts for a valid message. Few agents actually enforce these in the sense that a message whose header contents exceed one or more limits set there are

generally allowed to pass; they typically add any required fields that are missing, however.

Also, few agents that use messages as input, including Mail User Agents (MUAs) that actually display messages to users, verify that the input is valid before proceeding. Some popular open source filtering programs and some popular Mailing List Management (MLM) packages select either the first or last instance of a particular field name, such as From, to decide who sent a message. Absent strict enforcement of [MAIL], an attacker can craft a message with multiple instances of the same field fields if that attacker knows the filter will make a decision based on one but the user will be shown the others.

This situation is exacerbated when message validity is assessed, such as through enhanced authentication methods like DomainKeys Identified Mail [DKIM]. Such methods might cover one instance of a constrained field but not another, taking the wrong one as "good" or "safe". An MUA, for example could show the first of two From fields to an end user as "good" or "safe" while an authentication method actually only verified the second.

In attempting to counter this exposure, one of the following strategies can be used:

1. reject outright or refuse to process further any input message that does not conform to Section 3.6 of [MAIL];
2. remove or, in the case of an MUA, refuse to render any instances of a header field whose presence exceeds a limit prescribed in Section 3.6 of [MAIL] when generating its output;
3. where a field has a limited instance count, combine additional instances into a single instance carrying the same information as the multiple instances;
4. where a field can contain multiple distinct values (such as From) or is free-form text (such as Subject), combine them into a semantically identical single header field of the same name (see Section 7.5.1);
5. alter the name of any header field whose presence exceeds a limit prescribed in Section 3.6 of [MAIL] when generating its output so that later agents can produce a consistent result. Any alteration likely to cause the field to be ignored by downstream agents is acceptable. A common approach is to prefix the field names with a string such as "BAD-".

Selecting a mitigation action from the above list, or some other action, must consider the needs of the operator making the decision, and the nature of its user base.

7.5.1. Repeated Header Fields

There are some occasions where repeated fields are encountered where only one is expected. Two examples are presented. First:

```
From: reminders@example.com {1}
To: jqpublic@example.com {2}
Subject: Automatic Meeting Reminder {3}
Subject: 4pm Today -- Staff Meeting {4}
Date: Wed, 20 Oct 2010 08:00:00 -0700 {5}
```

```
Reminder of the staff meeting today in the small {6}
auditorium. Come early! {7}
```

The message above has two Subject fields, which is in violation of Section 3.6 of [MAIL]. A safe interpretation of this would be to treat it as though the two Subject field values were concatenated, so long as they are not identical, such as:

```
From: reminders@example.com {1}
To: jqpublic@example.com {2}
Subject: Automatic Meeting Reminder {3}
        4pm Today -- Staff Meeting {4}
Date: Wed, 20 Oct 2010 08:00:00 -0700 {5}
```

```
Reminder of the staff meeting today in the small {6}
auditorium. Come early! {7}
```

Second:

```
From: president@example.com {1}
From: vice-president@example.com {2}
To: jqpublic@example.com {3}
Subject: A note from the E-Team {4}
Date: Wed, 20 Oct 2010 08:00:00 -0700 {5}
```

```
This memo is to remind you of the corporate dress {6}
code. Attached you will find an updated copy of {7}
the policy. {8}
...
```

As with the first example, there is a violation in terms of the number of instances of the From field. A likely safe interpretation would be to combine these into a comma-separated address list in a

single From field:

```
From: president@example.com, {1}
      vice-president@example.com {2}
To: jqpublic@example.com {3}
Subject: A note from the E-Team {4}
Date: Wed, 20 Oct 2010 08:00:00 -0700 {5}
```

```
This memo is to remind you of the corporate dress {6}
code. Attached you will find an updated copy of {7}
the policy. {8}
...
```

7.5.2. Missing Header Fields

Similar to the previous section, there are messages seen in the wild that lack certain required header fields. In particular, [MAIL] requires that a From and Date field be present in all messages.

When presented with a message lacking these fields, the MTA might perform one of the following:

1. Make no changes
2. Add an instance of the missing field(s) using synthesized content based on data provided in other parts of the protocol

Option 2 is recommended for handling this case. Handling agents should add these for internal handling if they are missing, but should not add them to the external representation. The reason for this advice is that there are some filter modules that would consider the absence of such fields to be a condition warranting special treatment (for example, rejection), and thus the effectiveness of such modules would be stymied by an upstream filter adding them in a way visible to other components.

The synthesized fields should contain a best guess as to what should have been there; for From, the SMTP MAIL command's address can be used (if not null) or a placeholder address followed by an address literal (for example, unknown@[192.0.2.1]); for Date, a date extracted from a Received field is a reasonable choice.

One other important case to consider is a missing Message-Id field. An MTA that encounters a message missing this field should synthesize a valid one and add it to the external representation, since many deployed tools use the content of that field as a common unique message reference, so its absence inhibits correlation of message processing. Section 3.6.4 of [MAIL] describes advisable practise for

synthesizing the content of this field when it is absent, and establishes a requirement that it be globally unique.

7.5.3. Return-Path

A valid message will have exactly one Return-Path header field, as per Section 4.4 of [SMTP]. Should a message be encountered bearing more than one, all but the topmost one is to be disregarded, as it is most likely to have been added nearest to the mailbox that received that message.

7.6. Missing or Incorrect Charset Information

MIME provides the means to include textual material employing character sets ("charsets") other than US-ASCII. Such material is required to have an identified charset. Charset identification is done using a "charset" parameter in the Content-Type header field, a charset label within the MIME entity itself, or the charset can be implicitly specified by the Content-Type (see [CHARSET]).

It is unfortunately fairly common for required character set information to be missing or incorrect in textual MIME entities. As such, processing agents should perform basic sanity checks, such as:

- o US-ASCII contains bytes between 1 and 127 inclusive only (colloquially, "7-bit" data), so material including bytes outside of that range ("8-bit" data) is necessarily not US-ASCII. (See Section 2.3.1 of [MAIL].)
- o [UTF-8] has a very specific syntactic structure that other 8-bit charsets are unlikely to follow.
- o Null bytes (ASCII 0x00) are not allowed in either 7-bit or 8-bit data.
- o Not all 7-bit material is US-ASCII. The presence of the various escape sequences used for character switching can be used as an indication of the various charsets based on ISO/IEC 2022, such as those defined in [ISO-2022-CN], [ISO-2022-JP], and [ISO-2022-KR].

When a character set error is detected, processing agents should:

- a. apply heuristics to determine the most likely character set and, if successful, proceed using that information; or
- b. refuse to process the malformed MIME entity.

A null byte inside a textual MIME entity can cause typical string

processing functions to mis-identify the end of a string, which can be exploited to hide malicious content from analysis processes. Accordingly, null bytes require additional special handling.

A few null bytes in isolation is likely to be the result of poor message construction practices. Such nulls should be silently dropped.

Large numbers of null bytes are usually the result of binary material that is improperly encoded, improperly labeled, or both. Such material is likely to be damaged beyond the hope of recovery, so the best course of action is to refuse to process it.

Finally, the presence of null bytes may be used as indication of possible malicious intent.

7.7. Eight-Bit Data

Standards-compliant email messages do not contain any non-ASCII data without indicating that such content is present by means of published SMTP extensions. Absent that, MIME encodings are typically used to convert non-ASCII data to ASCII in a way that can be reversed by other handling agents or end users.

The best way to handle non-compliant 8bit material depends on its location.

Non-compliant 8bit material in MIME entity content should simply be processed as if the necessary SMTP extensions had been used to transfer the message. Note that improperly labeled 8bit material in textual MIME entities may require treatment as described in Section 7.6.

Non-compliant 8bit material in message or MIME entity header fields can be handled as follows:

- o Occurrences in unstructured text fields, comments, and phrases, can be converted into encoded-words (see [MIME3] if a likely character set can be determined). Alternatively, 8bit characters can be removed or replaced with some other character.
- o Occurrences in header fields whose syntax is unknown may be handled by dropping the field entirely or by removing/replacing the 8bit character as described above.
- o Occurrences in addresses are especially problematic. Agents supporting [EAI] may, if the 8bit material conforms to 8bit syntax, elect to treat the message as an EAI message and process

it accordingly. Otherwise, it is in most cases best to exclude the address from any sort of processing -- which may mean dropping it entirely -- since any attempt to fix it definitively is unlikely to be successful.

8. MIME Anomalies

The five-part set of MIME specifications includes a mechanism of message extensions for providing text in character sets other than ASCII, non-text attachments to messages, multi-part message bodies, and similar facilities.

Some anomalies with MIME-compliant generation are also common. This section discusses some of those and presents preferred mitigations.

8.1. Missing MIME-Version Field

Any message that uses [MIME] constructs is required to have a MIME-Version header field. Without it, the Content-Type and associated fields have no semantic meaning.

It is often observed that a message has complete MIME structure, yet lacks this header field. It is prudent to disregard this absence and conduct analysis of the message as if it were present, especially by agents attempting to identify malicious material.

Further, the absence of MIME-Version might be an indication of malicious intent, and extra scrutiny of the message may be warranted. Such omissions are not expected from compliant message generators.

8.2. Faulty Encodings

There have been a few different specifications of base64 in the past. The implementation defined in [MIME] instructs decoders to discard characters that are not part of the base64 alphabet. Other implementations consider an encoded body containing such characters to be completely invalid. Very early specifications of base64 (see [PEM], for example) allowed email-style comments within base64-encoded data.

The attack vector here involves constructing a base64 body whose meaning varies given different possible decodings. If a security analysis module wishes to be thorough, it should consider scanning the possible outputs of the known decoding dialects in an attempt to anticipate how the MUA will interpret the data.

9. Body Anomalies

9.1. Oversized Lines

A message containing a line of content that exceeds 998 characters plus the line terminator (1000 total) violates Section 2.1.1 of [MAIL]. Some handling agents may not look at content in a single line past the first 998 bytes, providing bad actors an opportunity to hide malicious content.

There is no specified way to handle such messages, other than to observe that they are non-compliant and reject them, or rewrite the oversized line such that the message is compliant.

To ensure long lines do not prevent analysis of potentially malicious data, handling agents are strongly encouraged to take one of the following actions:

1. Break such lines into multiple lines at a position that does not change the semantics of the text being thus altered. For example, breaking an oversized line such that a [URI] then spans two lines could inhibit the proper identification of that URI.
2. Rewrite the MIME part (or the entire message if not MIME) that contains the excessively long line using a content encoding that breaks the line in the transmission but would still result in the line being intact on decoding for presentation to the user. Both of the encodings declared in [MIME] can accomplish this.

10. Security Considerations

The discussions of the anomalies above and their prescribed solutions are themselves security considerations. The practises enumerated in this document are generally perceived as attempts to resolve security considerations that already exist rather than introducing new ones. However, some of the attacks described here may not have appeared in previous email specifications.

11. IANA Considerations

This document contains no actions for IANA.

[RFC Editor: Please remove this section prior to publication.]

12. References

12.1. Normative References

- [EMAIL-ARCH] Crocker, D., "Internet Mail Architecture", RFC 5598, July 2009.
- [MAIL] Resnick, P., "Internet Message Format", RFC 5322, October 2008.
- [MIME] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

12.2. Informative References

- [BINARYSMTP] Vaudreuil, G., "SMTP Service Extensions for Transmission of Large and Binary MIME Messages", RFC 3030, December 2000.
- [CHARSET] Melnikov, A. and J. Reschke, "Update to MIME regarding "charset" Parameter Handling in Textual Media Types", RFC 6657, July 2012.
- [DKIM] Crocker, D., Ed., Hansen, T., Ed., and M. Kucherawy, Ed., "DomainKeys Identified Mail (DKIM) Signatures", RFC 6376, September 2011.
- [DSN] Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 3464, January 2003.
- [EAI] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, February 2012.
- [ISO-2022-CN] Zhu, HF., Hu, DY., Wang, ZG., Kao, TC., Chang, WCH., and M. Crispin, "Chinese Character Encoding for Internet Messages", RFC 1922, March 1996.
- [ISO-2022-JP] Murai, J., Crispin, M., and E. van der Poel, "Japanese Character Encoding for Internet Messages", RFC 1468, June 1993.
- [ISO-2022-KR] Choi, U., Chon, K., and H. Park, "Korean Character Encoding for Internet Messages", RFC 1557, December 1993.
- [MIME3] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

- [PEM] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures", RFC 1113, August 1989.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -- Communication Layers", RFC 1122, October 1989.
- [RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, April 2001.
- [RFC733] Crocker, D., Vittal, J., Pogran, K., and D. Henderson, Jr., "Standard for the Format of Internet Text Messages", RFC 733, November 1977.
- [SMTP] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.
- [URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005.
- [UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, 2003.

Appendix A. RFC Editor Notes

[RFC Editor Note: This section can be removed before publication.]

I can't seem to figure out how to do this with xml2rfc, but the ISO-2022 reference above should contain the following URI:
http://www.iso.org/iso/catalogue_detail.htm?csnumber=22747

Appendix B. Acknowledgements

The author wishes to acknowledge the following for their review and constructive criticism of this proposal: Dave Cridland, Dave Crocker, Jim Galvin, Tony Hansen, John Levine, Franck Martin, Alexey Melnikov, and Timo Sirainen

Authors' Addresses

Murray S. Kucherawy

EMail: superuser@gmail.com

Gregory N. Shapiro

EMail: gshapiro@proofpoint.com

N. Freed

EMail: ned.freed@mrochek.com

Network Working Group
Internet Draft
Intended status: Standards Track
Expires: February 26, 2014

Paul E. Jones
Gonzalo Salgueiro
Cisco Systems
Michael B. Jones
Microsoft
Joseph Smarr
Google
August 26, 2013

WebFinger
draft-ietf-appsawg-webfinger-18.txt

Abstract

This specification defines the WebFinger protocol, which can be used to discover information about people or other entities on the Internet using standard HTTP methods. WebFinger discovers information for a URI that might not be usable as a locator otherwise, such as account or email URIs.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on February 26, 2014.

Copyright Notice

Copyright (c) 2013 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.....	2
2. Terminology.....	3
3. Example Uses of WebFinger.....	4
3.1. Identity Provider Discovery for OpenID Connect.....	4
3.2. Getting Author and Copyright Information for a Web Page...	5
4. WebFinger Protocol.....	6
4.1. Constructing the Query Component of the Request URI.....	7
4.2. Performing a WebFinger Query.....	7
4.3. The "rel" Parameter.....	8
4.4. The JSON Resource Descriptor (JRD).....	10
4.4.1. subject.....	10
4.4.2. aliases.....	10
4.4.3. properties.....	10
4.4.4. links.....	11
4.5. WebFinger and URIs.....	13
5. Cross-Origin Resource Sharing (CORS).....	13
6. Access Control.....	13
7. Hosted WebFinger Services.....	14
8. Definition of WebFinger Applications.....	15
8.1. Specification of the URI Scheme and URI.....	15
8.2. Host Resolution.....	15
8.3. Specification of Properties.....	16
8.4. Specification of Links.....	16
8.5. One URI, Multiple Applications.....	16
8.6. Registration of Link Relation Types and Properties.....	17
9. Security Considerations.....	17
9.1. Transport-Related Issues.....	17
9.2. User Privacy Considerations.....	17
9.3. Abuse Potential.....	18
9.4. Information Reliability.....	19
10. IANA Considerations.....	20
10.1. Well-Known URI.....	20
10.2. JSON Resource Descriptor (JRD) Media Type.....	20
10.3. Registering Link Relation Types.....	21
10.4. Establishment of the WebFinger Properties Registry.....	22
10.4.1. The Registration Template.....	22
10.4.2. The Registration Procedures.....	22
11. Acknowledgments.....	23
12. References.....	23
12.1. Normative References.....	23
12.2. Informative References.....	24
Author's Addresses.....	25

1. Introduction

WebFinger is used to discover information about people or other entities on the Internet that are identified by a URI [6] using standard Hypertext Transfer Protocol (HTTP) [2] methods over a secure transport [12]. A WebFinger resource returns a JavaScript Object

Notation (JSON) [5] object describing the entity that is queried. The JSON object is referred to as the JSON Resource Descriptor (JRD).

For a person, the kinds of information that might be discoverable via WebFinger include a personal profile address, identity service, telephone number, or preferred avatar. For other entities on the Internet, a WebFinger resource might return JRDs containing link relations [8] that enable a client to discover, for example, that a printer can print in color on A4 paper, the physical location of a server, or other static information.

Information returned via WebFinger might be for direct human consumption (e.g., looking up someone's phone number), or it might be used by systems to help carry out some operation (e.g., facilitate, with additional security mechanisms, logging into a web site by determining a user's identity service). The information is intended to be static in nature and, as such, WebFinger is not intended to be used to return dynamic information like the temperature of a CPU or the current toner level in a laser printer.

The WebFinger protocol is designed to be used across many applications. Applications that wish to utilize WebFinger will need to specify properties, titles, and link relation types that are appropriate for the application. Further, applications will need to define the appropriate URI scheme to utilize for the query target.

Use of WebFinger is illustrated in the examples in Section 3 and described more formally in Section 4. Section 8 describes how applications of WebFinger may be defined.

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 [1].

WebFinger makes heavy use of "Link Relations". A Link Relation is an attribute-and-value pair in which the attribute identifies the type of relationship between the linked entity or resource and the information specified in the value. In Web Linking [4], the link relation is represented using an HTTP entity-header of "Link", where the "rel" attribute specifies the type of relationship and the "href" attribute specifies the information that is linked to the entity or resource. In WebFinger, the same concept is represented using a JSON array of "links" objects, where each member named "rel" specifies the type of relationship and each member named "href" specifies the information that is linked to the entity or resource. Note that WebFinger narrows the scope of a link relation beyond what is defined for Web Linking by stipulating that the value of the "rel" member needs to be either a single IANA-registered link relation type [8] or a URI [6].

The use of URIs throughout this document refers to URIs following the syntax specified in Section 3 of RFC 3986 [6]. Relative URIs, having syntax following that of Section 4.2 or RFC 3986, are not used with WebFinger.

3. Example Uses of WebFinger

This section shows a few sample uses of WebFinger. Any application of WebFinger would be specified outside of this document, as described in Section 8. The examples in this section should be simple enough to understand without having seen the formal specifications of the applications.

3.1. Identity Provider Discovery for OpenID Connect

Suppose Carol wishes to authenticate with a web site she visits using OpenID Connect [15]. She would provide the web site with her OpenID Connect identifier, say carol@example.com. The visited web site would perform a WebFinger query looking for the OpenID Connect Provider. Since the site is interested in only one particular link relation, the WebFinger resource might utilize the "rel" parameter as described in Section 4.3:

```
GET /.well-known/webfinger?
    resource=acct%3Acarol%40example.com&
    rel=http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fissuer
HTTP/1.1
Host: example.com
```

The server might respond like this:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/jrd+json

{
  "subject" : "acct:carol@example.com",
  "links" :
  [
    {
      "rel" : "http://openid.net/specs/connect/1.0/issuer",
      "href" : "https://openid.example.com"
    }
  ]
}
```

Since the "rel" parameter only serves to filter the link relations returned by the resource, other name/value pairs in the response, including any aliases or properties, would be returned. Also, since support for the "rel" parameter is not guaranteed, the client must not assume the "links" array will contain only the requested link relation.

3.2. Getting Author and Copyright Information for a Web Page

Suppose an application is defined to retrieve metadata information about a web page URL, such as author and copyright information. To retrieve that information, the client can utilize WebFinger to issue a query for the specific URL. Suppose the URL of interest is `http://blog.example.com/article/id/314`. The client would issue a query similar to the following:

```
GET /.well-known/webfinger?  
    resource=http%3A%2F%2Fblog.example.com%2Farticle%2Fid%2F314  
HTTP/1.1  
Host: blog.example.com
```

The server might then reply in this way:

```
HTTP/1.1 200 OK  
Access-Control-Allow-Origin: *  
Content-Type: application/jrd+json  
  
{  
  "subject" : "http://blog.example.com/article/id/314",  
  "aliases" :  
  [  
    "http://blog.example.com/cool_new_thing",  
    "http://blog.example.com/steve/article/7"  
  ],  
  "properties" :  
  {  
    "http://blgx.example.net/ns/version" : "1.3",  
    "http://blgx.example.net/ns/ext" : null  
  },  
  "links" :  
  [  
    {  
      "rel" : "copyright",  
      "href" : "http://www.example.com/copyright"  
    },  
    {  
      "rel" : "author",  
      "href" : "http://blog.example.com/author/steve",  
      "titles" :  
      {  
        "en-us" : "The Magical World of Steve",  
        "fr" : "Le Monde Magique de Steve"  
      },  
      "properties" :  
      {  
        "http://example.com/role" : "editor"  
      }  
    }  
  ]  
}
```

```
    ]  
  }
```

In the above example, we see that the server returned a list of aliases, properties, and links related to the subject URL. The links contain references to information for each link relation type. For the author link, the server provided a reference to the author's blog, along with a title for the blog in two languages. The server also returned a single property related to the author, indicating the author's role as editor of the blog.

It is worth noting that, while the server returned just two links in the links array in this example, a server might return any number of links when queried.

4. WebFinger Protocol

The WebFinger protocol is used to request information about an entity identified by a query target (a URI). The client can optionally specify one or more link relation types for which it would like to receive information.

A WebFinger request is an HTTPS request to a WebFinger resource. A WebFinger resource is a well-known URI [3] using the HTTPS scheme, constructed along with the required query target and optional link relation types. WebFinger resources MUST NOT be served with any other URI scheme (such as HTTP).

A WebFinger resource is always given a query target, which is another URI that identifies the entity whose information is sought. GET requests to a WebFinger resource convey the query target in the "resource" parameter in the WebFinger URI's query string; see Section 4.1 for details.

The host to which a WebFinger query is issued is significant. If the query target contains a "host" portion (Section 3.2.2 of RFC 3986), then the host to which the WebFinger query is issued SHOULD be the same as the "host" portion of the query target, unless the client receives instructions through some out-of-band mechanism to send the query to another host. If the query target does not contain a "host" portion, then the client chooses a host to which it directs the query using additional information it has.

The path component of a WebFinger URI MUST be the well-known path `"/.well-known/webfinger"`. A WebFinger URI MUST contain a query component that encodes the query target and optional link relation types as specified in Section 4.1.

The WebFinger resource returns a JSON Resource Descriptor (JRD) as the resource representation to convey information about an entity on the Internet. Also, the Cross-Origin Resource Sharing (CORS) [7]

specification is utilized to facilitate queries made via a web browser.

4.1. Constructing the Query Component of the Request URI

A WebFinger URI MUST contain a query component (see Section 3.4 of RFC 3986). The query component MUST contain a "resource" parameter and MAY contain one or more "rel" parameters. The "resource" parameter MUST contain the query target (URI) and the "rel" parameters MUST contain encoded link relation types according to the encoding described in this section.

To construct the query component, the client performs the following steps. First, each parameter value is percent-encoded, as per Section 2.1 of RFC 3986. The encoding is done to conform to the query production in Section 3.4 of that specification, with the addition that any instances of the "=" and "&" characters within the parameter values are also percent-encoded. Next, the client constructs a string to be placed in the query component by concatenating the name of the first parameter together with an equal sign ("=") and the percent-encoded parameter value. For any subsequent parameters, the client appends an ampersand("&") to the string, the name of the next parameter, an equal sign, and the parameter value. The client MUST NOT insert any spaces while constructing the string. The order in which the client places each attribute-and-value pair within the query component does not matter in the interpretation of the query component.

4.2. Performing a WebFinger Query

A WebFinger client issues a query using the GET method to the well-known [3] resource identified by the URI whose path component is `"/.well-known/webfinger"` and whose query component MUST include the "resource" parameter exactly once and set to the value of the URI for which information is being sought.

If the "resource" parameter is absent or malformed, the WebFinger resource MUST indicate that the request is bad as per Section 10.4.1 of RFC 2616 [2].

If the "resource" parameter is a value for which the server has no information, the server MUST indicate that it was unable to match the request as per Section 10.4.5 of RFC 2616.

A client MUST query the WebFinger resource using HTTPS only. If the client determines that the resource has an invalid certificate, the resource returns a 4xx or 5xx status code, or the HTTPS connection cannot be established for any reason, then the client MUST accept that the WebFinger query has failed and MUST NOT attempt to reissue the WebFinger request using HTTP over a non-secure connection.

A WebFinger resource MUST return a JRD as the representation for the resource if the client requests no other supported format explicitly via the HTTP "Accept" header. The client MAY include the "Accept" header to indicate a desired representation; representations other than JRD might be defined in future specifications. The WebFinger resource MUST silently ignore any requested representations that it does not understand and support. The media type used for the JSON Resource Descriptor (JRD) is "application/jrd+json" (see Section 9.2).

The properties, titles, and link relation types returned by the server in a JRD might be varied and numerous. For example, the server might return information about a person's blog, vCard [14], avatar, OpenID Connect provider, RSS or ATOM feed, and so forth in a reply. Likewise, if a server has no information to provide it might return a JRD with an empty links array or no links array.

A WebFinger resource MAY redirect the client; if it does, the redirection MUST only be to an "https" URI and the client MUST perform certificate validation again when redirected.

A WebFinger resource can include cache validators in a response to enable conditional requests by the client and/or expiration times as per Section 13 of RFC 2616.

4.3. The "rel" Parameter

When issuing a request to a WebFinger resource, the client MAY utilize the "rel" parameter to request only a subset of the information that would otherwise be returned without the "rel" parameter. When the "rel" parameter is used and accepted, only the link relation types that match the link relation types provided via the "rel" parameter are included in the array of links returned in the JRD. If there are no matching link relation types defined for the resource, the "links" array in the JRD will either be absent or empty. All other information present in a resource descriptor remains present, even when "rel" is employed.

The "rel" parameter MAY be included multiple times in order to request multiple link relation types.

The purpose of the "rel" parameter is to return a subset of "link relation objects" (see Section 4.4.4) that would otherwise be returned in the resource descriptor. Use of the parameter might reduce processing requirements on either the client or server, and it might also reduce the bandwidth required to convey the partial resource descriptor, especially if there are numerous link relation values to convey for a given "resource" value. Note that if a client requests a particular link relation type for which the server has no information, the server MAY return a JRD with an empty links array or no links array.

WebFinger resources SHOULD support the "rel" parameter. If the resource does not support the "rel" parameter, it MUST ignore the parameter and process the request as if no "rel" parameter values were present.

The following example uses the "rel" parameter to request links for two link relation types:

```
GET /.well-known/webfinger?
    resource=acct%3Abob%40example.com&
    rel=http%3A%2F%2Fwebfinger.example%2Frel%2Fprofile-page&
    rel=http://webfinger.example/rel/businesscard HTTP/1.1
Host: example.com
```

In this example, the client requests the link relations of type "http://webfinger.example/rel/profile-page" and "http://webfinger.example/rel/businesscard". The server then responds with a message like this:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/jrd+json

{
  "subject" : "acct:bob@example.com",
  "aliases" :
  [
    "https://www.example.com/~bob/"
  ],
  "properties" :
  {
    "http://example.com/ns/role" : "employee"
  },
  "links" :
  [
    {
      "rel" : "http://webfinger.example/rel/profile-page",
      "href" : "https://www.example.com/~bob/"
    },
    {
      "rel" : "http://webfinger.example/rel/businesscard",
      "href" : "https://www.example.com/~bob/bob.vcf"
    }
  ]
}
```

As you can see in the response, the resource representation contains only the links of the types requested by the client and for which the server had information, but the other parts of the JRD are still present. Note also in the above example that the links returned in the links array all use HTTPS, which is important if the data indirectly obtained via WebFinger needs to be returned securely.

4.4. The JSON Resource Descriptor (JRD)

The JSON Resource Descriptor (JRD), originally introduced in RFC 6415 [16] and based on the Extensible Resource Descriptor (XRD) format [17], is a JSON object that comprises the following name/value pairs:

- o subject
- o aliases
- o properties
- o links

The member "subject" is a name/value pair whose value is a string, "aliases" is an array of strings, "properties" is an object comprising name/value pairs whose values are strings, and "links" is an array of objects that contain link relation information.

When processing a JRD, the client MUST ignore any unknown member and not treat the presence of an unknown member as an error.

Below, each of these members of the JRD is described in more detail.

4.4.1. subject

The value of the "subject" member is a URI that identifies the entity that the JRD describes.

The "subject" value returned by a WebFinger resource MAY differ from the value of the "resource" parameter used in the client's request. This might happen, for example, when the subject's identity changes (e.g., a user moves his or her account to another service) or when the resource prefers to express URIs in canonical form.

The "subject" member SHOULD be present in the JRD.

4.4.2. aliases

The "aliases" array is an array of zero or more URI strings that identify the same entity as the "subject" URI.

The "aliases" array is OPTIONAL in the JRD.

4.4.3. properties

The "properties" object comprises zero or more name/value pairs whose names are URIs (referred to as "property identifiers") and whose values are strings or null. Properties are used to convey additional information about the subject of the JRD. As an example, consider this use of "properties":

```
"properties" : { "http://webfinger.example/ns/name" : "Bob Smith" }
```

The "properties" member is OPTIONAL in the JRD.

4.4.4. links

The "links" array has any number of member objects, each of which represents a link [4]. Each of these link objects can have the following members:

- o rel
- o type
- o href
- o titles
- o properties

The "rel" and "href" members are strings representing the link's relation type and the target URI, respectively. The context of the link is the "subject" (see Section 4.4.1).

The "type" member is a string indicating what the media type of the result of dereferencing the link ought to be.

The order of elements in the "links" array MAY be interpreted as indicating an order of preference. Thus, if there are two or more link relations having the same "rel" value, the first link relation would indicate the user's preferred link.

The "links" array is OPTIONAL in the JRD.

Below, each of the members of the objects found in the "links" array is described in more detail. Each object in the "links" array, referred to as a "link relation object", is completely independent from any other object in the array; any requirement to include a given member in the link relation object refers only to that particular object.

4.4.4.1. rel

The value of the "rel" member is a string that is either a URI or a registered relation type [8] (see RFC 5988 [4]). The value of the "rel" member MUST contain exactly one URI or registered relation type. The URI or registered relation type identifies the type of the link relation.

The other members of the object have meaning only once the type of link relation is understood. In some instances, the link relation will have associated semantics enabling the client to query for other resources on the Internet. In other instances, the link relation will have associated semantics enabling the client to utilize the other members of the link relation object without fetching additional external resources.

URI link relation type values are compared using the "Simple String Comparison" algorithm of Section 6.2.1 of RFC 3986.

The "rel" member MUST be present in the link relation object.

4.4.4.2. type

The value of the "type" member is a string that indicates the media type [9] of the target resource (see RFC 6838 [10]).

The "type" member is OPTIONAL in the link relation object.

4.4.4.3. href

The value of the "href" member is a string that contains a URI pointing to the target resource.

The "href" member is OPTIONAL in the link relation object.

4.4.4.4. titles

The "titles" object comprises zero or more name/value pairs whose name is a language tag [11] or the string "und". The string is human-readable and describes the link relation. More than one title for the link relation MAY be provided for the benefit of users who utilize the link relation and, if used, a language identifier SHOULD be duly used as the name. If the language is unknown or unspecified, then the name is "und".

A JRD SHOULD NOT include more than one title identified with the same language tag (or "und") within the link relation object. Meaning is undefined if a link relation object includes more than one title named with the same language tag (or "und"), though this MUST NOT be treated as an error. A client MAY select whichever title or titles it wishes to utilize.

Here is an example of the titles object:

```
"titles" :
{
  "en-us" : "The Magical World of Steve",
  "fr" : "Le Monde Magique de Steve"
}
```

The "titles" member is OPTIONAL in the link relation object.

4.4.4.5. properties

The "properties" object within the link relation object comprises zero or more name/value pairs whose names are URIs (referred to as "property identifiers") and whose values are strings or null. Properties are used to convey additional information about the link relation. As an example, consider this use of "properties":

```
"properties" : { "http://webfinger.example/mail/port" : "993" }
```

The "properties" member is OPTIONAL in the link relation object.

4.5. WebFinger and URIs

WebFinger requests include a "resource" parameter (see Section 4.1) specifying the query target (URI) for which the client requests information. WebFinger is neutral regarding the scheme of such a URI: it could be an "acct" URI [18], an "http" or "https" URI, a "mailto" URI [19], or some other scheme.

5. Cross-Origin Resource Sharing (CORS)

WebFinger resources might not be accessible from a web browser due to "Same-Origin" policies. The current best practice is to make resources available to browsers through Cross-Origin Resource Sharing (CORS) [7], and servers MUST include the Access-Control-Allow-Origin HTTP header in responses. Servers SHOULD support the least restrictive setting by allowing any domain access to the WebFinger resource:

```
Access-Control-Allow-Origin: *
```

There are cases where defaulting to the least restrictive setting is not appropriate, for example a server on an intranet that provides sensitive company information SHOULD NOT allow CORS requests from any domain, as that could allow leaking of that sensitive information. A server that wishes to restrict access to information from external entities SHOULD use a more restrictive Access-Control-Allow-Origin header.

6. Access Control

As with all web resources, access to the WebFinger resource could require authentication. Further, failure to provide required credentials might result in the server forbidding access or providing a different response than had the client authenticated with the server.

Likewise, a WebFinger resource MAY provide different responses to different clients based on other factors, such as whether the client is inside or outside a corporate network. As a concrete example, a query performed on the internal corporate network might return link relations to employee pictures, whereas link relations for employee pictures might not be provided to external entities.

Further, link relations provided in a WebFinger resource representation might point to web resources that impose access restrictions. For example, the aforementioned corporate server may provide both internal and external entities with URIs to employee pictures, but further authentication might be required in order for the client to access the picture resources if the request comes from outside the corporate network.

The decisions made with respect to what set of link relations a WebFinger resource provides to one client versus another and what resources require further authentication, as well as the specific authentication mechanisms employed, are outside the scope of this document.

7. Hosted WebFinger Services

As with most services provided on the Internet, it is possible for a domain owner to utilize "hosted" WebFinger services. By way of example, a domain owner might control most aspects of their domain, but use a third-party hosting service for email. In the case of email, MX records identify mail servers for a domain. An MX record points to the mail server to which mail for the domain should be delivered. It does not matter to the sending mail server whether those MX records point to a server in the destination domain or a different domain.

Likewise, a domain owner might utilize the services of a third party to provide WebFinger services on behalf of its users. Just as a domain owner was required to insert MX records into DNS to allow for hosted email serves, the domain owner is required to redirect HTTP queries to its domain to allow for hosted WebFinger services.

When a query is issued to the WebFinger resource, the web server **MUST** return a response with a redirection status code that includes a Location header pointing to the location of the hosted WebFinger service URI. This WebFinger service URI does not need to point to the well-known WebFinger location on the hosting service provider server.

As an example, assume that example.com's WebFinger services are hosted by wf.example.net. Suppose a client issues a query for acct:alice@example.com like this:

```
GET /.well-known/webfinger?  
    resource=acct%3Aalice%40example.com HTTP/1.1  
Host: example.com
```

The server might respond with this:

```
HTTP/1.1 307 Temporary Redirect  
Access-Control-Allow-Origin: *  
Location: https://wf.example.net/example.com/webfinger?  
    resource=acct%3Aalice%40example.com
```

The client can then follow the redirection, re-issuing the request to the URI provided in the Location header. Note that the server will include any required URI parameters in the Location header value, which could be different than the URI parameters the client originally used.

8. Definition of WebFinger Applications

This specification details the protocol syntax used to query a domain for information about a URI, the syntax of the JSON Resource Descriptor (JRD) that is returned in response to that query, security requirements and considerations, hosted WebFinger services, various expected HTTP status codes, and so forth. However, this specification does not enumerate the various possible properties or link relation types that might be used in conjunction with WebFinger for a particular application, nor does it define what properties or link relation types one might expect to see in response to querying for a particular URI or URI scheme. Nonetheless, all of these unspecified elements are important in order to implement an interoperable application that utilizes the WebFinger protocol and MUST be specified in the relevant document(s) defining the particular application making use of the WebFinger protocol according to the procedures described in this section.

8.1. Specification of the URI Scheme and URI

Any application that uses WebFinger MUST specify the URI scheme(s) and, to the extent appropriate, what forms the URI(s) might take. For example, when querying for information about a user's account at some domain, it might make sense to specify the use of the acct URI scheme [18]. When trying to obtain the copyright information for a web page, it makes sense to specify the use of the web page URI (either http or https).

The examples in Sections 3.1 and 3.2 illustrate the use of different URI schemes with WebFinger applications. In the example in Section 3.1, WebFinger is used to retrieve information pertinent to OpenID Connect. In the example in Section 3.2, WebFinger is used to discover metadata information about a web page, including author and copyright information. Each of these applications of WebFinger needs to be fully specified to ensure interoperability.

8.2. Host Resolution

As explained in Section 4, the host to which a WebFinger query is issued is significant. In general, WebFinger applications would adhere to the procedures described in Section 4 in order to properly direct a WebFinger query.

However, some URI schemes do not have host portions and there might be some applications of WebFinger for which the host portion of a URI cannot or should not be utilized. In such instances, the application specification MUST clearly define the host resolution procedures, which might include provisioning a "default" host within the client to which queries are directed.

8.3. Specification of Properties

WebFinger defines both subject-specific properties (i.e., properties described in Section 4.4.3 that relate to the URI for which information is queried) and link-specific properties (see Section 4.4.4.5). This section refers to subject-specific properties.

Applications that utilize subject-specific properties MUST define the URIs used in identifying those properties, along with valid property values.

Consider this portion of the JRD found in the example in Section 3.2.

```
"properties" :
{
  "http://blgx.example.net/ns/version" : "1.3",
  "http://blgx.example.net/ns/ext" : null
}
```

Here, two properties are returned in the WebFinger response. Each of these would be defined in a WebFinger application specification. These two properties might be defined in the same WebFinger application specification or separately in different specifications. Since the latter is possible, it is important that WebFinger clients not assume that one property has any specific relationship with another property unless some relationship is explicitly defined in the particular WebFinger application specification.

8.4. Specification of Links

The links returned in a WebFinger response each comprise several pieces of information, some of which are optional (refer to Section 4.4.4). The WebFinger application specification MUST define each link and any values associated with a link, including the link relation type ("rel"), the expected media type ("type"), properties, and titles.

The target URI to which the link refers (i.e., the "href"), if present, would not normally be specified in an application specification. However, the URI scheme or any special characteristics of the URI would usually be specified. If a particular link does not require an external reference, then all of the semantics related to the use of that link MUST be defined within the application specification. Such links might rely only on properties or titles in the link to convey meaning.

8.5. One URI, Multiple Applications

It is important to be mindful of the fact that different WebFinger applications might specify the use of the same URI scheme and, in effect, the same URI for different purposes. That should not be a problem, since each of property identifier (see Sections 4.4.3 and

4.4.4.5) and link relation type would be uniquely defined for a specific application.

It should be noted that when a client requests information about a particular URI and receives a response with a number of different property identifiers or link relation types that the response is providing information about the URI without any particular semantics. How the client interprets the information SHOULD be in accordance with the particular application specification or set of specifications the client implements.

Any syntactically valid properties or links the client receives and that are not fully understood SHOULD be ignored and SHOULD NOT cause the client to report an error.

8.6. Registration of Link Relation Types and Properties

Application specifications MAY define a simple token as a link relation type for a link. In that case, the link relation type MUST be registered with IANA as specified in Sections 10.3.

Further, any defined properties MUST be registered with IANA as described in Section 10.4.

9. Security Considerations

9.1. Transport-Related Issues

Since this specification utilizes Cross-Origin Resource Sharing (CORS) [7], all of the security considerations applicable to CORS are also applicable to this specification.

The use of HTTPS is REQUIRED to ensure that information is not modified during transit. It should be appreciated that in environments where a web server is normally available, there exists the possibility that a compromised network might have its WebFinger resource operating on HTTPS replaced with one operating only over HTTP. As such, clients MUST NOT issue queries over a non-secure connection.

Clients MUST verify that the certificate used on an HTTPS connection is valid (as defined in [12]) and accept a response only if the certificate is valid.

9.2. User Privacy Considerations

Service providers and users should be aware that placing information on the Internet means that any user can access that information and WebFinger can be used to make it even easier to discover that information. While WebFinger can be an extremely useful tool for discovering one's avatar, blog, or other personal data, users should understand the risks, too.

Systems or services that expose personal data via WebFinger MUST provide an interface by which users can select which data elements are exposed through the WebFinger interface. For example, social networking sites might allow users to mark certain data as "public" and then utilize that marking as a means of determining what information to expose via WebFinger. The information published via WebFinger would thus comprise only the information marked as public by the user. Further, the user has the ability to remove information from publication via WebFinger by removing this marking.

WebFinger MUST NOT be used to provide any personal data unless publishing that data via WebFinger by the relevant service was explicitly authorized by the person whose information is being shared. Publishing one's personal data within an access-controlled or otherwise limited environment on the Internet does not equate to providing implicit authorization of further publication of that data via WebFinger.

The privacy and security concerns with publishing personal data via WebFinger are worth emphasizing again with respect to personal data that might reveal a user's current context (e.g., the user's location). The power of WebFinger comes from providing a single place where others can find pointers to information about a person, but service providers and users should be mindful of the nature of that information shared and the fact that it might be available for the entire world to see. Sharing location information, for example, would potentially put a person in danger from any individual who might seek to inflict harm on that person.

Users should be aware of how easily personal data one might publish can be used in unintended ways. In one study relevant to WebFinger-like services, Balduzzi et al. [20] took a large set of leaked email addresses and demonstrated a number of potential privacy concerns, including the ability to cross-correlate the same user's accounts over multiple social networks. The authors also describe potential mitigation strategies.

The easy access to user information via WebFinger was a design goal of the protocol, not a limitation. If one wishes to limit access to information available via WebFinger, such as WebFinger resources for use inside a corporate network, the network administrator needs to take necessary measures to limit access from outside the network. Using standard methods for securing web resources, network administrators do have the ability to control access to resources that might return sensitive information. Further, a server can be employed in such a way as to require authentication and prevent disclosure of information to unauthorized entities.

9.3. Abuse Potential

Service providers should be mindful of the potential for abuse using WebFinger.

As one example, one might query a WebFinger server only to discover whether a given URI is valid or not. With such a query, the person may deduce that an email identifier is valid, for example. Such an approach could help spammers maintain a current list of known email addresses and to discover new ones.

WebFinger could be used to associate a name or other personal data with an email address, allowing spammers to craft more convincing email messages. This might be of particular value in phishing attempts.

It is RECOMMENDED that implementers of WebFinger server software take steps to mitigate abuse, including malicious over-use of the server and harvesting of user information. Although there is no mechanism that can guarantee that publicly-accessible WebFinger databases won't be harvested, rate-limiting by IP address will prevent or at least dramatically slow harvest by private individuals without access to botnets or other distributed systems. The reason these mitigation strategies are not mandatory is that the correct choice of mitigation strategy (if any) depends greatly on the context. Implementers should not construe this as meaning that they do not need to consider whether to use a mitigation strategy, and, if so, what strategy to use.

WebFinger client developers should also be aware of potential abuse by spammers or those phishing for information about users. As an example, suppose a mail client was configured to automatically perform a WebFinger query on the sender of each received mail message. If a spammer sent an email using a unique identifier in the 'From' header, then when the WF query was performed the spammer would be able to associate the request with a particular user's email address. This would provide information to the spammer, including the user's IP address, the fact the user just checked email, what kind of WebFinger client the user utilized, and so on. For this reason, it is strongly advised that clients not perform WebFinger queries unless authorized by the user to do so.

9.4. Information Reliability

A WebFinger resource has no means of ensuring that information provided by a user is accurate. Likewise, neither the resource nor the client can be absolutely guaranteed that information has not been manipulated either at the server or along the communication path between the client and server. Use of HTTPS helps to address some concerns with manipulation of information along the communication path, but it clearly cannot address issues where the resource provided incorrect information, either due to being provided false information or due to malicious behavior on the part of the server administrator. As with any information service available on the Internet, users should be wary of information received from untrusted sources.

10. IANA Considerations

10.1. Well-Known URI

This specification registers the "webfinger" well-known URI in the Well-Known URI Registry as defined by [3].

URI suffix: webfinger

Change controller: IETF

Specification document(s): RFC XXXX

Related information: The query to the WebFinger resource will include one or more parameters in the query string; see Section 4.1 of RFCXXXX. Resources at this location are able to return a JSON Resource Descriptor (JRD) as described in Section 4.4 of RFCXXXX.

[RFC EDITOR: Please replace "XXXX" references in this section and the following section with the number for this RFC.]

10.2. JSON Resource Descriptor (JRD) Media Type

This specification registers the media type application/jrd+json for use with WebFinger in accordance with media type registration procedures defined in [10].

Type name: application

Subtype name: jrd+json

Required parameters: N/A

Optional parameters: N/A

In particular, because RFC 4627 already defines the character encoding for JSON, no "charset" parameter is used.

Encoding considerations: See RFC 6839, Section 3.1.

Security considerations:

The JSON Resource Descriptor (JRD) is a JavaScript Object Notation (JSON) object. It is a text format that must be parsed by entities that wish to utilize the format. Depending on the language and mechanism used to parse a JSON object, it is possible for an attacker to inject behavior into a running program. Therefore, care must be taken to properly parse a received JRD to ensure that only a valid JSON object is present and that no JavaScript or other code is injected or executed unexpectedly.

Interoperability considerations:

This media type is a JavaScript Object Notation (JSON) object and can be consumed by any software application that can consume JSON objects.

Published specification: RFC XXXX

Applications that use this media type:

The JSON Resource Descriptor (JRD) is used by the WebFinger protocol (RFC XXXX) to enable the exchange of information between a client and a WebFinger resource over HTTPS.

Fragment identifier considerations:

The syntax and semantics of fragment identifiers SHOULD be as specified for "application/json". (At publication of this document, there is no fragment identification syntax defined for "application/json".)

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): jrd

Macintosh file type code(s): N/A

Person & email address to contact for further information:

Paul E. Jones <paulej@packetizer.com>

Intended usage: COMMON

Restrictions on usage: N/A

Author: Paul E. Jones <paulej@packetizer.com>

Change controller:

IETF has change control over this registration.

Provisional registration? (standards tree only): N/A

10.3. Registering Link Relation Types

RFC 5988 established a Link Relation Type Registry that is re-used by WebFinger applications.

Link relation types used by WebFinger applications are registered in the Link Relations Type Registry as per the procedures of Section

6.2.1 of RFC 5988. The "Notes" entry for the registration SHOULD indicate if property values associated with the link relation type are registered in the WebFinger Properties registry with a link to the registry.

10.4. Establishment of the WebFinger Properties Registry

WebFinger utilizes URIs to identify properties of a subject or link and the associated values (see Section 8.3 and Section 8.6). This specification establishes a new "WebFinger Properties" registry to record property identifiers.

10.4.1. The Registration Template

The registration template for WebFinger properties is:

- o Property Identifier:
- o Link Type:
- o Description:
- o Reference:
- o Notes: [optional]

The "Property Identifier" must be a URI that identifies the property being registered.

The "Link Type" contains the name of a Link Relation Type with which this property identifier is used. If the property is a subject-specific property, then this field is specified as "N/A".

The "Description" is intended to explaining the purpose of the property.

The "Reference" field points to the specification that defines the registered property.

The optional "Notes" field is for conveying any useful information about the property that might be of value to implementers.

10.4.2. The Registration Procedures

The IETF has created a mailing list, webfinger@ietf.org, which can be used for public discussion of the WebFinger protocol and any applications that use it. Prior to registration of a WebFinger property, discussion on the mailing list is strongly encouraged. The IESG has appointed Designated Experts who will monitor the webfinger@ietf.org mailing list and review registrations.

A WebFinger property is registered with a Specification Required (see RFC 5226 [13]) after a review by the Designated Expert(s). The review is normally expected to take on the order of two to four weeks. However, the Designated Expert(s) may approve a registration prior to publication of a specification once the Designated Expert(s) are satisfied that such a specification will be published. In evaluating registration requests, the Designated Expert(s) should make an effort to avoid registering two different properties that have the same meaning. Where a proposed property is similar to an already-defined property, Designated Expert(s) should insist that enough text be included in the description or notes section of the template to sufficiently differentiate the new property from an existing one.

The registration procedure begins when a completed registration template (as defined above) sent to webfinger@ietf.org and iana@iana.org. IANA will track the review process and communicate the results to the registrant. The WebFinger mailing list provides an opportunity for community discussion and input, and the Designated Expert(s) may use that input to inform their review. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful if re-submitted.

The specification registering the WebFinger property MUST include the completed registration template shown above. Once the registration procedure concludes successfully, IANA creates or modifies the corresponding record in the "WebFinger Properties" registry.

11. Acknowledgments

This document has benefited from extensive discussion and review of many of the members of the APPSAWG working group. The authors would like to especially acknowledge the invaluable input of Eran Hammer-Lahav, Blaine Cook, Brad Fitzpatrick, Laurent-Walter Goix, Joe Clarke, Peter Saint-Andre, Dick Hardt, Tim Bray, James Snell, Melvin Carvalho, Evan Prodromou, Mark Nottingham, Elf Pavlik, Bjoern Hoehrmann, Subramanian Moonesamy, Joe Gregorio, John Bradley, and others that we have undoubtedly, but inadvertently, missed.

The authors would also like to express their gratitude to the chairs of APPSAWG, especially Salvatore Loreto for his assistance in shepherding this document. We also want to thank Barry Leiba and Pete Resnick, the Applications Area Directors, for their support and exhaustive reviews.

12. References

12.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [3] Nottingham, M., Hammer-Lahav, E., "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [4] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [5] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [6] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [7] Van Kesteren, A., "Cross-Origin Resource Sharing", W3C CORS <http://www.w3.org/TR/cors/>, July 2010.
- [8] IANA, "Link Relations", <http://www.iana.org/assignments/link-relations/>.
- [9] IANA, "MIME Media Types", <http://www.iana.org/assignments/media-types/index.html>.
- [10] Freed, N., Klensin, J., Hansen, T., "Media Type Specifications and Registration Procedures", RFC 6838, January 2013.
- [11] Phillips, A., Davis, M., "Tags for Identifying Languages", RFC 5646, January 2009.
- [12] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [13] Narten, T. and H. Alvestrand, "Guidelines for Writing an, IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

12.2. Informative References

- [14] Perreault, S., "vCard Format Specification", RFC 6350, August 2011.
- [15] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, "OpenID Connect Messages 1.0", July 2013, http://openid.net/specs/openid-connect-messages-1_0.html.
- [16] Hammer-Lahav, E. and Cook, B., "Web Host Metadata", RFC 6415, October 2011.
- [17] Hammer-Lahav, E. and W. Norris, "Extensible Resource Descriptor (XRD) Version 1.0", <http://docs.oasis-open.org/xri/xrd/v1.0/xrd-1.0.html>.

- [18] Saint-Andre, P., "The 'acct' URI Scheme", draft-ietf-appsawg-acct-uri-06, July 2013.
- [19] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, October 2010.
- [20] Balduzzi, Marco, et al., "Abusing social networks for automated user profiling", Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 2010, https://www.eurecom.fr/en/publication/3042/download/rs-publi-3042_1.pdf.

Author's Addresses

Paul E. Jones
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA

Phone: +1 919 476 2048
Email: paulej@packetizer.com
IM: <xmpp:paulej@packetizer.com>

Gonzalo Salgueiro
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA

Phone: +1 919 392 3266
Email: gsalguei@cisco.com
IM: <xmpp:gsalguei@cisco.com>

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: <http://self-issued.info/>

Joseph Smarr
Google

Email: jsmarr@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 1, 2018

A. Newton
ARIN
P. Cordell
Codalogic
September 28, 2017

A Language for Rules Describing JSON Content
draft-newton-json-content-rules-09

Abstract

This document describes a language for specifying and testing the expected content of JSON structures found in JSON-using protocols, software, and processes.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://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 April 1, 2018.

Copyright Notice

Copyright (c) 2017 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 (<https://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
1.1. A First Example: Specifying Content	3
1.2. A Second Example: Testing Content	3
2. Overview of the Language	5
3. Lines and Comments	7
4. Rules	8
4.1. Rule Names and Assignments	8
4.2. Annotations	9
4.3. Starting Points and Root Rules	10
4.4. Type Specifications	10
4.5. Primitive Specifications	12
4.5.1. Numbers, Booleans and Null	12
4.5.2. Strings	13
4.6. Any Type	16
4.7. Member Specifications	16
4.8. Object Specifications	16
4.9. Array Specifications	19
4.9.1. Unordered Array Specifications	21
4.10. Group Specifications	21
4.11. Ordered and Unordered Groups in Arrays	22
4.12. Sequence and Choice Combinations in Array, Object, and Group Specifications	22
4.13. Repetition in Array, Object, and Group Specifications . .	23
4.14. Negating Evaluation	25
5. Directives	26
5.1. jcr-version	26
5.2. ruleset-id	27
5.3. import	27
6. Tips and Tricks	28
6.1. Any Member with Any Value	28
6.2. Lists of Values	29
6.3. Groups in Arrays	29
6.4. Groups in Objects	30
6.5. Group Rules as Macros	31
6.6. Object Mixins	31
6.7. Subordinate Dependencies	31
7. Implementation Status	32
7.1. JCR Validator	32
7.2. Codalogic JCR Parser	33
7.3. JCR Java	33
8. ABNF Syntax	33
9. Acknowledgements	39
10. References	39
10.1. Normative References	39
10.2. Infomative References	40
10.3. URIs	40

Appendix A. Co-Constraints	40
Appendix B. Testing Against JSON Content Rules	41
B.1. Locally Overriding Rules	41
B.2. Rule Callbacks	42
Appendix C. Changes from -07 and -08	42
Authors' Addresses	42

1. Introduction

This document describes JSON Content Rules (JCR), a language for specifying and testing the interchange of data in JSON [RFC7159] format used by computer protocols and processes. The syntax of JCR is not JSON but is "JSON-like", possessing the conciseness and utility that has made JSON popular.

1.1. A First Example: Specifying Content

The following JSON data describes a JSON object with two members, "line-count" and "word-count", each containing an integer.

```
{ "line-count" : 3426, "word-count" : 27886 }
```

Figure 1

This is also JCR that describes a JSON object with a member named "line-count" that is an integer that is exactly 3426 and a member named "word-count" that is an integer that is exactly 27886.

For a protocol specification, it is probably more useful to specify that each member is any integer and not specific, exact integers:

```
{ "line-count" : integer, "word-count" : integer }
```

Figure 2

Since line counts and word counts should be either zero or a positive integer, the specification may be further narrowed:

```
{ "line-count" : 0.. , "word-count" : 0.. }
```

Figure 3

1.2. A Second Example: Testing Content

Building on the first example, this second example describes the same object but with the addition of another member, "file-name".

```
{
  "file-name" : "rfc7159.txt",
  "line-count" : 3426,
  "word-count" : 27886
}
```

Figure 4

The following JCR describes objects like it.

```
{
  "file-name" : string,
  "line-count" : 0..,
  "word-count" : 0..
}
```

Figure 5

For the purposes of writing a protocol specification, JCR may be broken down into named rules to reduce complexity and to enable reuse. The following example takes the JCR from above and rewrites the members as named rules.

```
{
  $fn,
  $lc,
  $wc
}

$fn = "file-name" : string
$lc = "line-count" : 0..
$wc = "word-count" : 0..
```

Figure 6

With each member specified as a named rule, software testers can override them locally for specific test cases. In the following example, the named rules are locally overridden for the test case where the file name is "rfc4627.txt".

```
$fn = "file-name" : "rfc4627.txt"
$lc = "line-count" : 2102
$wc = "word-count" : 16714
```

Figure 7

In this example, the protocol specification describes the JSON object in general and an implementation overrides the rules for testing specific cases.

All figures used in this specification are available here [1].

2. Overview of the Language

JCR is composed of rules (as the name suggests). A collection of rules that is processed together is a ruleset. Rulesets may also contain comments, blank lines, and directives that apply to the processing of a ruleset.

Rules are composed of two parts, an optional rule name and a rule specification. A rule specification can be either a type specification or a member specification. A member specification consists of a member name specification and a type specification.

A type specification is used to specify constraints on a superset of a JSON value (e.g. number / string / object / array etc.). In addition to defining primitive types (such as string or integer), array types, and object types, type specifications may define the JCR specific concept of group types.

Type specifications corresponding to arrays, objects and groups may be composed of other rule specifications.

A member specification is used to specify constraints on a JSON member (i.e. members of a JSON object).

Rules with rule name assignments may be referenced in place of type specifications and member specifications.

Rules may be defined across line boundaries and there is no line continuation syntax.

Any rule consisting only of a type specification is considered a root rule. Unless otherwise specified, all the root rules of a ruleset are evaluated against a JSON instance or document.

Putting it all together, Figure 9 describes the JSON in Figure 8.

Example JSON shamelessly lifted from RFC 4627

```
{
  "Image": {
    "Width": 800,
    "Height": 600,
    "Title": "View from 15th Floor",
    "Thumbnail": {
      "Url": "http://www.example.com/image/481989943",
      "Height": 125,
      "Width": 100
    },
    "IDs": [116, 943, 234, 38793]
  }
}
```

Figure 8

Rules describing Figure 8

```
; the root of the JSON instance is an object
; this root rule describes that object
{

    ; the object specification contains
    ; one member specification
    "Image" : {

        ; $width and $height are defined below
        $width,
        $height,

        ; "Title" member specification
        "Title" :string,

        ; "Thumbnail" member specification, which
        ; defines an object
        "Thumbnail": {

            ; $width and $height are re-used again
            $width, $height,

            "Url" :uri
        },

        ; "IDs" member that is an array of
        ; one ore more integers
        "IDs" : [ integer * ]

    }
}

; The definitions of the rules $width and $height
$width  = "Width" : 0..1280
$height = "Height" : 0..1024
```

Figure 9

3. Lines and Comments

There is no statement terminator and therefore no need for a line continuation syntax. Rules may be defined across line boundaries. Blank lines are allowed.

Comments are the same as comments in ABNF [RFC4234]. They start with a semi-colon (';') and continue to the end of the line.

4. Rules

Rules have two main components, an optional rule name assignment and a type or member specification.

Type specifications define arrays, objects, etc... of JSON and may reference other rules using rule names. Most type specifications can be defined with repetitions for specifying the frequency of the type being defined. In addition to the type specifications describing JSON types, there is an additional group specification for grouping types.

Member specifications define members of JSON objects, and are composed of a member name specification and either a type specification or a rule name referencing a type specification.

Rules may also contain annotations which may affect the evaluation of all or part of a rule. Rules without a rule name assignment are considered root rules, though rules with a rule name assignment can be considered a root rule with the appropriate annotation.

Type specifications, depending on their type, can contain zero or more other specifications or rule names. For example, an object specification might contain multiple member specifications or rule names that resolve to member specifications or a mixture of member specifications and rule names. For the purposes of this document, specifications and rule names composing other specifications are called subordinate components.

4.1. Rule Names and Assignments

Rule names are signified with the dollar character ('\$'), which is not part of the rule name itself. Rule names have two components, an optional ruleset identifier alias and a local rule name.

Local rule names must start with an alphabetic character (a-z,A-Z) and must contain only alphabetic characters, numeric characters, the hyphen character ('-') and the underscore character ('_'). Local rule names are case sensitive, and must be unique within a ruleset (that is, no two rule name assignments may use the same local rule name).

Ruleset identifier aliases enable referencing rules from another ruleset. They are not allowed in rule name assignments, and only found in rule names referencing other rules. Ruleset identifiers

must start with an alphabetic character and contain no whitespace. Ruleset identifiers are case sensitive. Simple use cases of JCR will most likely not use ruleset identifiers.

In Figure 10 below, "http://ietf.org/rfcYYYY.JCR" and "http://ietf.org/rfcXXXX.JCR" are ruleset identifiers and "rfcXXXX" is a ruleset identifier alias.

```
# ruleset-id http://ietf.org/rfcYYYY.JCR
# import http://ietf.org/rfcXXXX.JCR as rfcXXXX
$my_encodings = ( "mythic" | "magic" )
$all_encodings = ( $rfcXXXX.encodings | $my_encodings )
```

Figure 10

There are two forms of rule name assignments: assignments of primitive types and assignments of all other types. Rule name assignments to primitive type specifications separate the rule name from the type specification with the character sequence '=: ', whereas rule name assignments for all other type specifications only require the separation using the '=' character.

```
;rule name assignments for primitive types
$foo          =: "foo"
$some_string =: string

;rule name assignments for arrays
$bar = [ integer, integer, integer ]

;rule name assignment for objects
$bob = { "bar" : $bar, "foo" : $foo }
```

Figure 11

This is the one little "gotcha" in JCR. This syntax is necessary so that JCR parsers may readily distinguish between rule name assignments involving string and regular expressions primitive types and member names of member specifications.

4.2. Annotations

Annotations may appear before a rule name assignment, before a type or member specification, or before a rule name contained within a type specification. In each place, there may be zero or more annotations. Each annotation begins with the character sequence "@{" and ends with "}". The following is an example of a type specification with the not annotation (explained in Section 4.14):

```
@{not} [ "fruits", "vegatables" ]
```

Figure 12

This specification defines the annotations "root", "not", and "unordered", but other annotations may be defined for other purposes.

4.3. Starting Points and Root Rules

Evaluation of a JSON instance or document against a ruleset begins with the evaluation of a root rule or set of root rules. If no root rule (or rules) is specified locally at runtime, the set of root rules specified in the ruleset are evaluated. The order of evaluation is undefined.

The set of root rules specified in a ruleset is composed of all rules without a rule name assignment and all rules annotated with the "@{root}" annotation.

The "@{root}" annotation may either appear before a rule name assignment or before a type definition. It is ignored if present before referenced rule name inside of a type specification.

4.4. Type Specifications

The syntax of each type of type specifications varies depending on the type:

```
; primitive types can be string
; or number literals
; or number ranges
"foo"
2
1..10

; primitive types can also be more generalized types
string
integer

; primitive type rules may be named
$my_int =: 12

; member specifications consist of a member name
; followed by a colon and then followed by another
; type specification or a rule name
; (example shown with a rule name assignment)
$mem1 = "bar" : "baz"
$mem2 = "fizz" : $my_int

; member names may either be quoted strings
; or regular expressions
; (example shown with a rule name assignment)
$mem3 = /^dev[0-9]$/ : 0..4096

; object specifications start and end with "curly braces"
; object specifications contain zero
; or more member specifications
; or rule names which reference a member specification
{ $mem1, "foo" : "fuzz", "fizz" : $my_int }

; array specifications start and end with square brackets
; array specifications contain zero
; or more non-member type specifications
[ 1, 2, 3, $my_int ]

; finally, group specifications start and end with parenthesis
; groups contain other type specifications
( [ integer, integer], $rule1 )
$rule1 = [ string, string ]
```

Figure 13

4.5. Primitive Specifications

Primitive type specifications define content for JSON numbers, booleans, strings, and null.

4.5.1. Numbers, Booleans and Null

The rules for booleans and null are the simplest and take the following forms:

```
true
false
boolean
null
```

Figure 14

Rules for numbers can specify the number be either an integer or floating point number:

```
integer
float
double
```

Figure 15

The keyword 'float' represents a single precision IEEE-754 floating point number represented in decimal. The keyword 'double' represents a double precision IEEE-754 floating point number represented in decimal format.

Numbers may also be specified as an absolute value or a range of possible values, where a range may be specified using a minimum, maximum, or both:

```
n
n..m
..m
n..
n.f
n.f..m.f
..m.f
n.f..
```

Figure 16

When specifying a minimum and a maximum, both must either be an integer or a floating point number. Thus to specify a floating point

number between zero and ten a definition of the following form is used:

0.0..10.0

Figure 17

Integers may also be specified as ranges using bit lengths preceded by the 'int' or 'uint' words (i.e. 'int8', 'uint16'). The 'int' prefix specifies the integer as being signed whereas the 'uint' prefix specifies the integer as being unsigned.

```
; 0..255
uint8

; -32768..32767
int16

; 0..65535
uint16

; -9223372036854775808..9223372036854775807
int64

; 0..18446744073709551615
uint64
```

Figure 18

4.5.2. Strings

JCR provides a large number of data types to define the contents of JSON strings. Generically, a string may be specified using the word 'string'. String literals may be specified using a double quote character followed by the literal content followed by another double quote. And regular expressions may be specified by enclosing a regular expression within the forward slash ('/') character.

```
; any string
string

; a string literal
"she sells sea shells"

; a regular expression
/^she sells .*/
```

Figure 19

Regular expressions are not implicitly anchored and therefore must be explicitly anchored if necessary.

A string can be specified as a URI [RFC3986] using the word 'uri', but also may be more narrowly scoped to a URI of a specific scheme. Specific URI schemes are specified with the word 'uri' followed by two period characters ('..') followed by the URI scheme.

```
; any URI
uri

;a URI narrowed for an HTTPS uri
uri..https
```

Figure 20

IP addresses may be specified with either the word 'ipv4' for IPv4 addresses [RFC1166] or the word 'ipv6' for IPv6 addresses [RFC5952]. Fully qualified A-label and U-label domain names may be specified with the words 'fqdn' and 'idn'.

Dates and time can be specified as formats found in RFC 3339 [RFC3339]. The word 'date' corresponds to the full-date ABNF rule, the word 'time' corresponds to the full-time ABNF rule, and the word 'datetime' corresponds to the 'date-time' ABNF rule.

Email addresses formatted according to RFC 5322 [RFC5322] may be specified using the 'email' word, and E.123 phone numbers may be specified using the word 'phone'.

```
;IP addresses
ipv4
ipv6
ipaddr

;domain names
fqdn
idn

; RFC 3339 full-date
date
; RFC 3339 full-time
time
; RFC 3339 date-time
datetime

; RFC 5322 email address
email

; phone number
phone
```

Figure 21

Binary data can be specified in string form using the encodings specified in RFC 4648 [RFC4648]. The word 'hex' corresponds to base16, while 'base32', 'base32hex', 'base64', and 'base64url' correspond with their RFC 4648 counterparts accordingly.

```
; RFC 4648 base16
hex

; RFC 4648 base32
base32

; RFC 4648 base32hex
base32hex

; RFC 4648 base64
base64

; RFC 4648 base64url
base64url
```

Figure 22

4.6. Any Type

It is possible to specify that a value can be of any type allowable by JSON using the word 'any'. The 'any' type specifies any primitive type, array, or object.

4.7. Member Specifications

Member specifications define members of JSON objects. Unlike other type specifications, member specifications cannot be root rules and must be part of an object specification or preceded by a rule name assignment.

Member specifications consist of a member name specification followed by a colon character (':') followed by either a subordinate component, which is either a rule name or a primitive, object, array, or group specification. Member name specifications can be given either as a quoted string using double quotes or as a regular expression using forward slash ('/') characters. Regular expressions are not implicitly anchored and therefore must have explicit anchors if needed.

```
;member name will exactly match "locationURI"
$location_uri = "locationURI" : uri

;member name will match "eth0", "eth1", ... "eth9"
$iface_mappings = /^eth[0-9]$/ : ipv4
```

Figure 23

4.8. Object Specifications

Object specifications define JSON objects and are composed of zero or more subordinate components, each of which can be either a rule name, member specification, or group specification. The subordinate components are enclosed at the start with a left curly brace character ('{') and at the end with a right curly brace character ('}').

Evaluation of the subordinate components of object specifications is as follows:

- o No order is implied for the members of the object being evaluated.
- o Subordinate components of the object specification are evaluated in the order they appear.

- o Each member of the object being evaluated can only match one subordinate component.
- o Any members not matched against a subordinate component are ignored.

The following examples illustrate matching of JSON objects to JCR object specifications.

As order is not implied for the members of objects under evaluation, the following rule will match the JSON in Figure 25 and Figure 26.

```
{ "locationUri" : uri, "statusCode" : integer }
```

Figure 24

```
{ "locationUri" : "http://example.com", "statusCode" : 200 }
```

Figure 25

```
{ "statusCode" : 200, "locationUri" : "http://example.com" }
```

Figure 26

Because subordinate components of an object specification are evaluated in the order in which they are specified (i.e. left to right, top to bottom) and object members can only match one subordinate component of an object specification, the rule o1 below will not match against the JSON in Figure 28 but the rule o2 below will match it.

```

; zero or more members that match "p0", "p1", etc
; and a member that matches "p1"
$o1 = { /^p\d+$/ : integer *, "p1" : integer }

; a member that matches "p1" and
; zero or more members that match "p0", "p1", etc
$o2 = { "p1" : integer, /^p\d+$/ : integer * }

```

The first subordinate of rule o1 specifies that an object can have zero or more members (that is the meaning of "*", see Section 4.13) where the member name is the letter 'p' followed by a number (e.g. "p0", "p1", "p2"), and the second rule specifies a member with the exact member name of "p1". Rule o2 has the exact same member specifications but in the opposite order. Figure 28 does not match rule o1 because all of the members match the first subordinate rule leaving none to match the second subordinate rule. However, rule o2 does match because the first subordinate rule matches only one member of the JSON object allowing the second subordinate rule to match the other member of the JSON object.

Figure 27

```
{ "p0" : 1, "p1" : 2 }
```

Figure 28

As stated above, members of objects which do not match a rule are ignored. The reason for this validation model is due to the nature of the typical access model to JSON objects in many programming languages, where members of the object are obtained by referencing the member name. Therefore extra members may exist without harm.

However, some specifications may need to restrict the members of a JSON object to a known set. To construct a rule specifying that no extra members are expected, the @*{not}* annotation (see Section 4.14) may be used with a "match-all" regular expression as the last subordinate component of the object specification.

The following rule will match the JSON object in Figure 30 but will not match the JSON object in Figure 31.

```
{ "foo" : 1, "bar" : 2, @{not} // : any + }
```

Figure 29

```
{ "foo" : 1, "bar" : 2 }
```

Figure 30

```
{ "foo" : 1, "bar" : 2, "baz" : 3 }
```

Figure 31

This works because subordinate components are evaluated in the order they appear in the object rule, and the last component accepts any member with any type but fails to validate if one or more of those components are found due to the @{not} annotation.

4.9. Array Specifications

Array specifications define JSON arrays and are composed of zero or more subordinate components, each of which can either be a rule name or a primitive, array, object or group specification. The subordinate components are enclosed at the start with a left square brace character ('[') and at the end with a right square brace character (']').

Evaluation of the subordinate components of array specifications is as follows:

- o The order of array items is implied unless the @{unordered} annotation is present.
- o Subordinate components of the array specification are evaluated in the order they appear.
- o Each item of the array being evaluated can only match one subordinate component of the array specification.
- o If any items of the array are not matched, then the array does not match the array specification.

These rules are further explained in the examples below.

```
[ 0..1024, 0..980 ]
```

Figure 32

Unlike object specifications, order is implied in array specifications by default. That is, the first subordinate component will match the first element of the array, the second subordinate component will match the second element of the array, and so on.

Take for example the following ruleset:

```
; the first element of the array is to be a string
; the second element of the array is to be an integer
$a1 = [ string, integer ]

; the first element of the array is to be an integer
; the second element of the array is to be a string
$a2 = [ integer, string ]
```

Figure 33

It defines two rules, a1 and a2. The array in the following JSON will not match a1, but will match a2.

```
[ 24, "Bob Smurd" ]
```

Figure 34

If an array has more elements than can be matched from the array specification, the array does not match the array specification. Or stated differently, an array with unmatched elements does not validate. Using the example array rule a2 from above, the following array does not match because the last element of the array does not match any subordinate component:

```
[ 24, "Bob Smurd", "http://example.com/bob_smurd" ]
```

Figure 35

To allow an array to contain any value after guaranteeing that it contains the necessary items, the last subordinate component of the array specification should accept any item:


```
; the first element of the array is to be an integer
; the second element of the array is to be a string
; anything else can follow
$a3 = [ integer, string, any * ]
```

The JSON array in Figure 35 will validate against the a3 rule in this example.

Figure 36

4.9.1. Unordered Array Specifications

Array specifications can be made to behave in a similar fashion to object specifications with regard to the order of matching with the `@{unordered}` annotation.

In the ruleset below, a1 and a2 have the same subordinate components given in the same order. a2 is annotated with the `@{unordered}` annotation.

```
$a1 = [ string, integer ]
$a2 = @{unordered} [ string, integer ]
```

Figure 37

The JSON array below does not match a1 but does match a2.

```
[ 24, "Bob Smurd" ]
```

Figure 38

Like ordered array specifications, the subordinate components in an unordered array specification are evaluated in the order they are specified. The difference is that they need not match an element of the array in the same position as given in the array specification.

Finally, like ordered array specifications, unordered array specifications also require that all elements of the array be matched by a subordinate component. If the array has more elements than can be matched, the array does not match the array specification.

4.10. Group Specifications

Unlike the other type specifications, group specifications have no direct tie with JSON syntax. Group specifications simply group together their subordinate components. Group specifications enclose one or more subordinate components with the parenthesis characters.

Group specifications and any nesting of group specifications, must conform to the allowable set of type specifications of the type specifications in which they are contained. For example, a group specification inside of an array specification may not contain a member specification since member specifications are not allowed as direct subordinates of array specifications (arrays contain values, not object members in JSON). Likewise, a group specification referenced inside an object specification must only contain member specifications (JSON objects may only contain object members).

The following is an example of a group specification:

```
$the_bradys = [ $parents, $children ]  
  
$children = ( "Greg", "Marsha", "Bobby", "Jan" )  
  
$parents = ( "Mike", "Carol" )
```

Figure 39

Like the subordinate components of array and object specifications, the subordinate components of a group specification are evaluated in the order they appear.

4.11. Ordered and Unordered Groups in Arrays

Section 4.9.1 specifies that arrays can be evaluated by the order of the items in the array or can be evaluated without order. Section 4.10 specifies that arrays may have group rules as subordinate components.

The evaluation of a group specification inside an array specification inherits the ordering property of the array specification. If the array specification is unordered, then the items of the group specification are also considered to be unordered. And if the array specification is ordered, then the items of the group specification are also considered to be ordered.

4.12. Sequence and Choice Combinations in Array, Object, and Group Specifications

Combinations of subordinate components in array, object, and group specifications can be specified as either a sequence ("and") or a choice ("or"). A sequence is a subordinate component followed by the comma character (',') followed by another subordinate component. A choice is a subordinate component followed by a pipe character ('|') followed by another subordinate component.

```
; sequence ("and")
[ "this" , "that" ]

; choice ("or")
[ "this" | "that" ]
```

Figure 40

Sequence and choice combinations cannot be mixed, and group specifications must be used to explicitly declare precedence between a sequence and a choice. Therefore, the following is illegal:

```
[ "this", "that" | "the_other" ]
```

Figure 41

The example above should be expressed as:

```
[ "this", ( "that" | "the_other" ) ]
```

Figure 42

NOTE: A future specification will clarify the choice ('|') operation as inclusive or, exclusive or ("xor") or otherwise. At present readers should assume the choice ('|') operator is an inclusive or. However, for objects and unordered arrays that is not ideal, nor is xor. We are in the process of defining an algorithm to "rewrite" choices of rules for use with inclusive or which is more suitable for the data model of JSON.

4.13. Repetition in Array, Object, and Group Specifications

Evaluation of subordinate components in array, object, and group specifications may be succeeded by a repetition expression denoting how many times the subordinate component should be evaluated. Repetition expressions are specified using a Kleene symbol ('?', '+', or '*') or with the '*' symbol succeeded by specific minimum and/or maximum values, each being non-negative integers. Repetition expressions may also be appended with a step expression, which is the '%' symbol followed by a positive integer.

When no repetition expression is present, both the minimum and maximum are 1.

A minimum and maximum can be expressed by giving the minimum followed by two period characters ('..') followed by the maximum, with either the minimum or maximum being optional. When the minimum is not

explicitly specified, it is assumed to be zero. When the maximum is not explicitly specified, it is assumed to be positive infinity.

```

; exactly 2 octets
$word = [ $octet *2 ]
$octet =: int8

; 1 to 13 name servers
[ $name_servers *1..13 ]
$name_servers =: fqdn

; 0 to 99 ethernet addresses
{ /^eth.*/ : $mac_addr *..99 }
$mac_addr =: hex

; four or more bytes
[ $octet *4.. ]

```

Figure 43

The allowable Kleene operators are the question mark character ('?') which specifies zero or one (i.e. optional), the plus character ('+') which specifies one or more, and the asterisk character ('*') which specifies zero or more.

```

; age is optional
{ "name" : string, "age" : integer ? }

; zero or more errors
$error_set = ( string * )

; 1 or more integer values
[ integer + ]

```

Figure 44

A repetition step expression may follow a minimum to maximum expression or the zero or more Kleene operator or the one or more Kleene operator.

- o When the repetition step follows a minimum to maximum expression or the zero or more Kleene operator ('*'), it specifies that the total number of repetitions present in the JSON instance being validated minus the minimum repetition value must be a multiple of the repetition step (e.g. the total repetitions minus the minimum repetition value must be divisible by the step value with a remainder of zero).

- o When the repetition step follows a one or more Kleene operator ('+'), the minimum repetition value is set equal to the repetition step value and the total number of repetitions minus the step value must be a multiple of the repetition step value.

The following is an example for repetition steps in repetition expressions.

```

; there must be at least 2 name servers
; there may be no more than 12 name servers
; there must be an even number of name servers
; e.g. 2,4,6,8,10,12
[ $name_servers *2..12%2 ]
$name_servers =: fqdn

; minimum is zero
; maximum is 100
; must be an even number
{ /^eth.*/ : $mac_addr *..100%2 }
$mac_addr =: hex

; at least 32 octets
; must be in groups of 16
; e.g. 32, 48, 64 etc
[ $octet *32..%16 ]
$octet =: int8

; if there are to be error sets,
; their number must be divisible by 4
; e.g. 0, 4, 8, 12 etc
$error_set = ( string *%4 )

; Throws of a pair of dice must be divisible by 2
; e.g. 2, 4, 6 etc
$dice_throws = ( 1..6 +%2 )

```

Figure 45

4.14. Negating Evaluation

The evaluation of a rule can be changed with the `@{not}` annotation. With this annotation, a rule that would otherwise match does not, and a rule that would not have matched does.

```
; match anything that isn't the integer 2
$not_two = [ @{"not"} 2 ]

; error if one of the status values is "fail"
$status = @{"not"} @{"unordered"} [ "fail", string * ]
```

Figure 46

5. Directives

Directives modify the processing of a ruleset. There are two forms of the directive, the single line directive and the multi-line directive.

Single line directives appear on their own line in a ruleset, begin with a hash character ('#') and are terminated by the end of the line. They take the following form:

```
# directive_name parameter_1 parameter_2 ...
```

Figure 47

Multi-line directives also appear on their own lines, but may span multiple lines. They begin with the character sequence "{#" and end with "}". They take the following form:

```
{# directive_name
  parameter_1 parameter_2
  parameter_3
  ...
}
```

Figure 48

This specification defines the directives "jcr-version", "ruleset-id", and "import", but other directives may be defined.

5.1. jcr-version

This directive declares that the ruleset complies with a specific version of this standard. The version is expressed as a major integer followed by a period followed by a minor integer.

```
# jcr-version 0.7
```

Figure 49

The major.minor number signifying compliance with this document is "0.7". Upon publication of this specification as an IETF proposed standard, it will be "1.0".

```
# jcr-version 1.0
```

Figure 50

Ruleset authors are advised to place this directive as the first line of a ruleset.

This directive may have optional extension identifiers following the version number. Each extension identifier is preceded by the plus ('+') character and separated by white space. The format of extension identifiers is specific to the extension, but it is recommended that they are terminated by a version number.

```
# jcr-version 1.0 +co-constraints-1.2 +jcr-doc-1.0
```

Figure 51

5.2. ruleset-id

This directive identifies a ruleset to rule processors. It takes the form:

```
# ruleset-id identifier
```

Figure 52

An identifier can be a URL (e.g. `http://example.com/foo`), an inverted domain name (e.g. `com.example.foo`) or any other form that conforms to the JCR ABNF syntax that a ruleset author deems appropriate. To a JCR processor the identifier is treated as an opaque, case-sensitive string.

5.3. import

The import directive specifies that another ruleset is to have its rules evaluated in addition to the ruleset where the directive appears.

The following is an example:

```
# import http://example.com/rfc9999 as rfc9999
```

Figure 53

The rule names of the ruleset to be imported may be referenced by prepending the alias followed by a period character ('.') followed by the rule name (i.e. "alias.name"). To continue the example above, if the ruleset at <http://example.com/rfc9999> were to have a rule named 'encoding', rules in the ruleset importing it can refer to that rule as 'rfc9999.encoding'.

6. Tips and Tricks

6.1. Any Member with Any Value

Because member names may be specified with regular expressions, it is possible to construct a member rule that matches any member name. As an example, the following defines an object with a member with any name that has a value that is a string:

```
{ // : string }
```

Figure 54

The JSON below matches the above rule.

```
{ "foo" : "bar" }
```

Figure 55

Likewise, the JSON below also matches the same rule.

```
{ "fuzz" : "bazz" }
```

Figure 56

Constructing an object with a member of any name with any type would therefore take the form:

```
{ // : any }
```

Figure 57

The above rule matches not only the two JSON objects above, but the JSON object below.

```
{ "fuzz" : 1234 }
```

Figure 58

6.2. Lists of Values

Group specifications may be used to create enumerated lists of primitive data types, because primitive specifications may contain a group specification, which may have multiple primitive specifications. Because a primitive specification must resolve to a single data type, the group specification must only contain choice combinations.

Consider the following examples:

```
; either an IPv4 or IPv6 address
$address =: ( ipv4 | ipv6 )

; allowable fruits
$fruits =: ( "apple" | "banana" | "pear" )
```

Figure 59

6.3. Groups in Arrays

Groups may be a subordinate component of array specifications:

```
[ ( ipv4 | ipv6 ), integer ]
```

Figure 60

Unlike primitive specifications, subordinate group specifications in array specifications may have sequence combinations and contain any type specification.

```
; a group in an array
[ ( $first_name, $middle_name ?, $last_name ), $age ]

; a group referenced from an array
[ $name, $age ]
$name = ( $first_name, $middle_name ?, $last_name )

$first_name =: string
$middle_name =: string
$last_name =: string
$age =: 0..
```

Figure 61

6.4. Groups in Objects

Groups may be a subordinate component of object specifications: Subordinate group specifications in object specifications may have sequence combinations but must only contain member specifications.

```

; a group in an object
{ ( $title, $date, $author ), $paragraph + }

; a group referenced from an object
{ $front_matter, $paragraph + }
$front_matter = ( $title, $date, $author )

$title = "title" : string
$date = "date" : date
$author = "author" : [ string * ]
$paragraph = /p[0-9]*/ : string

```

Figure 62

NOTE: A future specification will clarify the choice ('|') operation as inclusive or, exclusive or ("xor") or otherwise. At present readers should assume the choice ('|') operator is an inclusive or. We are in the process of defining an algorithm to "rewrite" choices of rules for use with inclusive or which is more suitable for the data model of JSON. Such a change will impact the guidance given below.

When using groups to use both sequences and choices of member specifications, consideration must be given to the processing of object specifications where by unmatched member specifications are ignored (see Figure 23).

A casual reading of this rule might lead a reader to believe that the JSON object in Figure 64 would not match, however it does because the extra member (either "foo" or "baz") is not matched but is ignored.

```
{ "bar":string, ( "foo":integer | "baz":string ) }
```

Figure 63

```
{ "bar":"thing", "foo":2, "baz": "thingy" }
```

Figure 64

The rule in Figure 63 must be modified to either match all extra rules, as in Figure 65, or the logic of the rules must be rewritten

to explicitly negate the presence of the unwanted members, as in Figure 66.

```
{ "bar":string, ( "foo":integer | "baz":string ), @{not} //:any + }
```

Figure 65

```
{ "bar":string,
  ( ( "foo":integer , @{not} "baz":string ) |
    ( "baz":string , @{not} "foo":integer )
  ) }
```

Figure 66

6.5. Group Rules as Macros

The syntax for group specifications accommodates one or more subordinate components and a repetition expression for each. Other than grouping multiple rules, a group specification can be used as a macro definition for a single rule.

```
$paragraphs = ( /p[0-9]*/ : string + )
```

Figure 67

6.6. Object Mixins

Group rules can be used to create object mixins, a pattern for writing data models similar in style to object derivation in some programming languages. In the example in below, both obj1 and obj2 have a members "foo" and "fob" with obj1 having the additional member "bar" and obj2 having the additional member "baz".

```
$mixin_group = ( "foo" : integer, "fob" : uri )

$obj1 = { $mixin_group, "bar" : string }

$obj2 = { $mixin_group, "baz" : string }
```

Figure 68

6.7. Subordinate Dependencies

In object and array specifications, there may be situations in which it is necessary to condition the existence of a subordinate component on the existence of a sibling subordinate component. In other words, example_two should only be evaluated if example_one evaluates positively. Or put another way, a member of an object or an item of

an array may be present only on the condition that another member or item is present.

In the following example, the `referrer_uri` member can only be present if the `location_uri` member is present.

```
    ; $referrer_uri can only be present if
    ; $location_uri is present
    { ( $location_uri, $referrer_uri? )? }

    $location_uri = "locationURI" : uri
    $referrer_uri = "referrerURI" : uri
```

Figure 69

7. Implementation Status

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 [RFC7492] . 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 [RFC7492] , "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".

7.1. JCR Validator

The JCR Validator, written in Ruby, currently implements all portions of this specification, and has been used extensively to prototype various aspects of JCR under consideration. It's development has gone hand-in-hand with this specification.

This software is primarily produced by the American Registry for Internet Numbers (ARIN) and freely distributable under the ISC license.

Source code for this software is available on GitHub at [<https://github.com/arineng/jcrvalidator>](https://github.com/arineng/jcrvalidator). This software is also easily obtained as a Ruby Gem through the Ruby Gem system.

7.2. Codalogic JCR Parser

The Codalogic JCR Parser is a C++ implementation of a JCR parsing engine, and is a work in progress. It is targeted for the Windows platform.

This software is produced by Codalogic Ltd and freely distributable under the Gnu LGPL v3 license.

Source code is available on GitHub at [<https://github.com/codalogic/cl-jcr-parser>](https://github.com/codalogic/cl-jcr-parser).

7.3. JCR Java

JCR Java is a work in progress and currently only implements the parsing of JCR rulesets according to the ABNF using a custom parsing framework.

This software is produced by the American Registry for Internet Numbers (ARIN) and freely distributable under the MIT license.

Source code is available on BitBucket at [<https://bitbucket.org/aneutron_1998/jcr_java>](https://bitbucket.org/aneutron_1998/jcr_java).

8. ABNF Syntax

The following ABNF describes the syntax for JSON Content Rules. A text file containing these ABNF rules can be downloaded from [JCR_ABNF].

```
jcr                = *( sp-cmt / directive / root-rule / rule )

sp-cmt              = spaces / comment
spaces              = 1*( WSP / CR / LF )
DSPs                = ; Directive spaces
                    1*WSP /      ; When in one-line directive
                    1*sp-cmt   ; When in muti-line directive
comment             = ";" *comment-char comment-end-char
comment-char        = HTAB / %x20-10FFFF
                    ; Any char other than CR / LF
comment-end-char    = CR / LF

directive           = "#" (one-line-directive / multi-line-directive)
one-line-directive  = [ DSPs ]
```

```

        (directive-def / one-line-tbd-directive-d)
        *WSP eol
multi-line-directive = "{" *sp-cmt
        ( directive-def /
          multi-line-tbd-directive-d )
        *sp-cmt "}"
directive-def        = jcr-version-d / ruleset-id-d / import-d
jcr-version-d        = jcr-version-kw DSPs major-version
                      "." minor-version
                      *( DSPs "+" [ DSPs ] extension-id )
major-version        = non-neg-integer
minor-version        = non-neg-integer
extension-id         = ALPHA *not-space
ruleset-id-d         = ruleset-id-kw DSPs ruleset-id
import-d             = import-kw DSPs ruleset-id
                      [ DSPs as-kw DSPs ruleset-id-alias ]
ruleset-id           = ALPHA *not-space
not-space            = %x21-10FFFF
ruleset-id-alias     = name
one-line-tbd-directive-d = directive-name
                      [ WSP one-line-directive-parameters ]
directive-name       = name
one-line-directive-parameters = *not-eol
not-eol              = HTAB / %x20-10FFFF
eol                  = CR / LF
multi-line-tbd-directive-d = directive-name
                      [ 1*sp-cmt multi-line-directive-parameters ]
multi-line-directive-parameters = multi-line-parameters
multi-line-parameters = *(comment / q-string / regex /
                          not-multi-line-special)
not-multi-line-special = spaces / %x21 / %x23-2E / %x30-3A /
                          %x3C-7C / %x7E-10FFFF ; not " , / , ; or }

root-rule            = value-rule / group-rule

rule                 = annotations "$" rule-name *sp-cmt
                      "=" *sp-cmt rule-def

rule-name            = name
target-rule-name     = annotations "$"
                      [ ruleset-id-alias "." ]
                      rule-name
name                 = ALPHA *( ALPHA / DIGIT / "-" / "_" )

rule-def             = member-rule / type-designator rule-def-type-rule /
                      array-rule / object-rule / group-rule /
                      target-rule-name
type-designator      = type-kw 1*sp-cmt / ":" *sp-cmt

```

```
rule-def-type-rule = value-rule / type-choice
value-rule         = primitive-rule / array-rule / object-rule
member-rule        = annotations
                    member-name-spec *sp-cmt ":" *sp-cmt type-rule
member-name-spec   = regex / q-string
type-rule          = value-rule / type-choice / target-rule-name
type-choice        = annotations "(" type-choice-items
                    *( choice-combiner type-choice-items ) ")"
explicit-type-choice = type-designator type-choice
type-choice-items  = *sp-cmt ( type-choice / type-rule ) *sp-cmt

annotations        = *( "@" *sp-cmt annotation-set *sp-cmt ")"
                    *sp-cmt )
annotation-set      = not-annotation / unordered-annotation /
                    root-annotation / tbd-annotation
not-annotation      = not-kw
unordered-annotation = unordered-kw
root-annotation     = root-kw
tbd-annotation      = annotation-name [ spaces annotation-parameters ]
annotation-name     = name
annotation-parameters = multi-line-parameters

primitive-rule      = annotations primitive-def
primitive-def       = string-type / string-range / string-value /
                    null-type / boolean-type / true-value /
                    false-value / double-type / float-type /
                    float-range / float-value /
                    integer-type / integer-range / integer-value /
                    sized-int-type / sized-uint-type / ipv4-type /
                    ipv6-type / ipaddr-type / fqdn-type / idn-type /
                    uri-type / phone-type / email-type /
                    datetime-type / date-type / time-type /
                    hex-type / base32hex-type / base32-type /
                    base64url-type / base64-type / any
null-type           = null-kw
boolean-type        = boolean-kw
true-value          = true-kw
false-value         = false-kw
string-type         = string-kw
string-value        = q-string
string-range        = regex
double-type         = double-kw
float-type          = float-kw
float-range         = float-min ".." [ float-max ] / ".." float-max
float-min           = float
float-max           = float
float-value         = float
integer-type        = integer-kw
```

```

integer-range      = integer-min ".." [ integer-max ] /
                    ".." integer-max
integer-min        = integer
integer-max        = integer
integer-value      = integer
sized-int-type     = int-kw pos-integer
sized-uint-type    = uint-kw pos-integer
ipv4-type          = ipv4-kw
ipv6-type          = ipv6-kw
ipaddr-type        = ipaddr-kw
fqdn-type          = fqdn-kw
idn-type           = idn-kw
uri-type           = uri-kw [ ".." uri-scheme ]
phone-type         = phone-kw
email-type         = email-kw
datetime-type      = datetime-kw
date-type          = date-kw
time-type          = time-kw
hex-type           = hex-kw
base32hex-type     = base32hex-kw
base32-type        = base32-kw
base64url-type     = base64url-kw
base64-type        = base64-kw
any                = any-kw

object-rule        = annotations "{" *sp-cmt
                        [ object-items *sp-cmt ] }"
object-items       = object-item [ 1*( sequence-combiner object-item ) /
                                1*( choice-combiner object-item ) ]
object-item        = object-item-types *sp-cmt [ repetition *sp-cmt ]
object-item-types  = object-group / member-rule / target-rule-name
object-group       = annotations "(" *sp-cmt [ object-items *sp-cmt ] ")"

array-rule         = annotations "[" *sp-cmt [ array-items *sp-cmt ] "]"
array-items        = array-item [ 1*( sequence-combiner array-item ) /
                                1*( choice-combiner array-item ) ]
array-item         = array-item-types *sp-cmt [ repetition *sp-cmt ]
array-item-types   = array-group / type-rule / explicit-type-choice
array-group        = annotations "(" *sp-cmt [ array-items *sp-cmt ] ")"

group-rule         = annotations "(" *sp-cmt [ group-items *sp-cmt ] ")"
group-items        = group-item [ 1*( sequence-combiner group-item ) /
                                1*( choice-combiner group-item ) ]
group-item         = group-item-types *sp-cmt [ repetition *sp-cmt ]
group-item-types   = group-group / member-rule /
                    type-rule / explicit-type-choice
group-group        = group-rule

```



```

sequence-combiner = "," *sp-cmt
choice-combiner   = "|" *sp-cmt

repetition        = optional / one-or-more /
                    repetition-range / zero-or-more
optional          = "?"
one-or-more       = "+" [ repetition-step ]
zero-or-more      = "*" [ repetition-step ]
repetition-range  = "*" *sp-cmt (
                    min-max-repetition / min-repetition /
                    max-repetition / specific-repetition )
min-max-repetition = min-repeat ".." max-repeat
                    [ repetition-step ]
min-repetition    = min-repeat ".." [ repetition-step ]
max-repetition    = ".." max-repeat [ repetition-step ]
min-repeat        = non-neg-integer
max-repeat        = non-neg-integer
specific-repetition = non-neg-integer
repetition-step   = "%" step-size
step-size         = non-neg-integer

integer           = "0" / ["-"] pos-integer
non-neg-integer   = "0" / pos-integer
pos-integer       = digit1-9 *DIGIT

float             = [ minus ] int frac [ exp ]
                    ; From RFC 7159 except 'frac' required
minus            = %x2D ; -
plus             = %x2B ; +
int              = zero / ( digit1-9 *DIGIT )
digit1-9         = %x31-39 ; 1-9
frac             = decimal-point 1*DIGIT
decimal-point    = %x2E ; .
exp              = e [ minus / plus ] 1*DIGIT
e                = %x65 / %x45 ; e E
zero             = %x30 ; 0

q-string         = quotation-mark *char quotation-mark
                    ; From RFC 7159
char             = unescaped /
                    escape (
                    %x22 / ; " quotation mark U+0022
                    %x5C / ; \ reverse solidus U+005C
                    %x2F / ; / solidus U+002F
                    %x62 / ; b backspace U+0008
                    %x66 / ; f form feed U+000C
                    %x6E / ; n line feed U+000A
                    %x72 / ; r carriage return U+000D

```

```

        %x74 /           ; t      tab           U+0009
        %x75 4HEXDIG )   ; uXXXX           U+XXXX
escape      = %x5C           ; \
quotation-mark = %x22       ; "
unescaped   = %x20-21 / %x23-5B / %x5D-10FFFF

regex       = "/" *( escape "/" / not-slash ) "/"
              [ regex-modifiers ]
not-slash   = HTAB / CR / LF / %x20-2E / %x30-10FFFF
              ; Any char except "/"
regex-modifiers = *( "i" / "s" / "x" )

uri-scheme  = 1*ALPHA

;; Keywords
any-kw      = %x61.6E.79           ; "any"
as-kw       = %x61.73              ; "as"
base32-kw   = %x62.61.73.65.33.32  ; "base32"
base32hex-kw = %x62.61.73.65.33.32.68.65.78 ; "base32hex"
base64-kw   = %x62.61.73.65.36.34  ; "base64"
base64url-kw = %x62.61.73.65.36.34.75.72.6C ; "base64url"
boolean-kw  = %x62.6F.6F.6C.65.61.6E ; "boolean"
date-kw     = %x64.61.74.65        ; "date"
datetime-kw = %x64.61.74.65.74.69.6D.65 ; "datetime"
double-kw   = %x64.6F.75.62.6C.65  ; "double"
email-kw    = %x65.6D.61.69.6C     ; "email"
false-kw    = %x66.61.6C.73.65     ; "false"
float-kw    = %x66.6C.6F.61.74     ; "float"
fqdn-kw     = %x66.71.64.6E        ; "fqdn"
hex-kw      = %x68.65.78           ; "hex"
idn-kw      = %x69.64.6E           ; "idn"
import-kw   = %x69.6D.70.6F.72.74  ; "import"
int-kw      = %x69.6E.74           ; "int"
integer-kw  = %x69.6E.74.65.67.65.72 ; "integer"
ipaddr-kw   = %x69.70.61.64.64.72  ; "ipaddr"
ipv4-kw     = %x69.70.76.34        ; "ipv4"
ipv6-kw     = %x69.70.76.36        ; "ipv6"
jcr-version-kw = %x6A.63.72.2D.76.65.72.73.69.6F.6E ; "jcr-version"
not-kw      = %x6E.6F.74           ; "not"
null-kw     = %x6E.75.6C.6C        ; "null"
phone-kw    = %x70.68.6F.6E.65     ; "phone"
root-kw     = %x72.6F.6F.74        ; "root"
ruleset-id-kw = %x72.75.6C.65.73.65.74.2D.69.64 ; "ruleset-id"
string-kw   = %x73.74.72.69.6E.67 ; "string"
time-kw     = %x74.69.6D.65        ; "time"
true-kw     = %x74.72.75.65        ; "true"
type-kw     = %x74.79.70.65        ; "type"
uint-kw     = %x75.69.6E.74        ; "uint"

```

```

unordered-kw    = %x75.6E.6F.72.64.65.72.65.64    ; "unordered"
uri-kw          = %x75.72.69                      ; "uri"

;; Referenced RFC 5234 Core Rules
ALPHA           = %x41-5A / %x61-7A    ; A-Z / a-z
CR              = %x0D                ; carriage return
DIGIT           = %x30-39             ; 0-9
HEXDIG          = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
HTAB            = %x09                ; horizontal tab
LF              = %x0A                ; linefeed
SP              = %x20                ; space
WSP             = SP / HTAB          ; white space

```

Figure 70: ABNF for JSON Content Rules

9. Acknowledgements

John Cowan, Andrew Biggs, Paul Kyzivat and Paul Jones provided feedback and suggestions which led to many changes in the syntax.

10. References

10.1. Normative References

- [JCR_ABNF] Newton, A. and P. Cordell, "ABNF for JSON Content Rules", <<https://raw.githubusercontent.com/arineneng/jcr/master/jcr-abnf.txt>>.
- [RFC1166] Kirkpatrick, S., Stahl, M., and M. Recker, "Internet numbers", RFC 1166, DOI 10.17487/RFC1166, July 1990, <<https://www.rfc-editor.org/info/rfc1166>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, DOI 10.17487/RFC4234, October 2005, <<https://www.rfc-editor.org/info/rfc4234>>.

- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <<https://www.rfc-editor.org/info/rfc5322>>.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <<https://www.rfc-editor.org/info/rfc5952>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.

10.2. Informative References

- [I-D.cordell-jcr-co-constraints] Cordell, P. and A. Newton, "Co-Constraints for JSON Content Rules", draft-cordell-jcr-co-constraints-00 (work in progress), March 2016.
- [RFC7492] Bhatia, M., Zhang, D., and M. Jethanandani, "Analysis of Bidirectional Forwarding Detection (BFD) Security According to the Keying and Authentication for Routing Protocols (KARP) Design Guidelines", RFC 7492, DOI 10.17487/RFC7492, March 2015, <<https://www.rfc-editor.org/info/rfc7492>>.

10.3. URIs

- [1] <https://github.com/arineng/jcr/tree/master/figs>

Appendix A. Co-Constraints

This specification defines a small set of annotations and directives for JCR, yet the syntax is extensible allowing for other annotations and directives. [I-D.cordell-jcr-co-constraints] ("Co-Constraints for JCR") defines further annotations and directives which define more detailed constraints on JSON messages, including co-constraints (constraining parts of JSON message based on another part of a JSON message).

Appendix B. Testing Against JSON Content Rules

One aspect of JCR that differentiates it from other format schema languages are the mechanisms helpful to developers for taking a formal specification, such as that found in an RFC, and evolving it into unit tests, which are essential to producing quality protocol implementations.

B.1. Locally Overriding Rules

As mentioned in the introduction, one tool for testing would be the ability to locally override named rules. As an example, consider the following rule which defines an array of strings.

```
$statuses = [ string * ]
```

Figure 71

Consider the specification where this rule is found does not define the values but references an extensible list of possible values updated independently of the specification, such as in an IANA registry.

If a software developer desired to test a specific situation in which the array must at least contain the status "accepted", the rules from the specification could be used and the statuses rule could be explicitly overridden locally as:

This rule will evaluate positively with the JSON in Figure 73

```
$statuses = @{unordered} [ "accepted", string * ]
```

Figure 72

```
[ "submitted", "validated", "accepted" ]
```

Figure 73

Alternatively, the developer may need to ensure that the status "denied" should not be present in the array:

This rule will fail to evaluate the JSON in Figure 75 thus signaling a problem.

```
$statuses = @{unordered} @{not} [ "denied" + , string * ]
```

Figure 74

```
[ "submitted", "validated", "denied" ]
```

Figure 75

B.2. Rule Callbacks

In many testing scenarios, the evaluation of rules may become more complex than that which can be expressed in JCR, sometimes involving variables and interdependencies which can only be expressed in a programming language.

A JCR processor may provide a mechanism for the execution of local functions or methods based on the name of a rule being evaluated. Such a mechanism could pass to the function the data to be evaluated, and that function could return to the processor the result of evaluating the data in the function.

Appendix C. Changes from -07 and -08

This revision of the document makes no substantive changes to any parts of the specification. Some of the ABNF has been updated to more correctly allow group rules, and other small change have been made to the ABNF to make it simpler.

Authors' Addresses

Andrew Lee Newton
American Registry for Internet Numbers
PO Box 232290
Centreville, VA 20120
US

Email: andy@arin.net
URI: <http://www.arin.net>

Pete Cordell
Codalogic
PO Box 30
Ipswich IP5 2WY
UK

Email: pete.cordell@codalogic.com
URI: <http://www.codalogic.com>