Network Working Group
Internet-Draft
Intended status: Experimental
Expires: September 11, 2019

E. Kline
Google Japan
K. Duleba
Z. Szamonek
Google Switzerland GmbH
March 10, 2019

A Format for Self-published IP Geolocation Feeds
draft-google-self-published-geofeeds-03

Abstract

This document records a format whereby a network operator can publish a mapping of IP address prefixes to simplified geolocation information, colloquially termed a geolocation "feed". Interested parties can poll and parse these feeds to update or merge with other geolocation data sources and procedures. This format intentionally only allows specifying coarse level location.

Some technical organizations operating networks that move from one conference location to the next have already experimentally published small geolocation feeds. At least one consumer (Google) has incorporated these ad hoc feeds into a geolocation data pipeline, and is using it to allow ISPs to inform them where the prefixes live.

[RFC Ed - Please remove publication: The IETF Meeting network currently publishes a feed in this format at: https://noc.ietf.org/geo/google.csv -- this has significantly cut down on the number of "Gah! Why does the network believe I'm in Montreal, that was last meeting! How am I supposed to find a pub?!" complaints. A number of other meeting networks, including RIPE and ICANN publish this information as well, see below.]

[Ed note: Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication.]

[This document is being collaborated on in Github at: https://github.com/google/self-published-geo . The most recent version of the document, open issues, etc should all be available here. The authors (gratefully) accept pull requests]

Status of This Memo

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

This Internet-Draft will expire on September 11, 2019.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivation

Providers of services over the Internet have grown to depend on best-effort geolocation information to improve the user experience. Locality information can aid in directing traffic to the nearest serving location, inferring likely native language, and providing additional context for services involving search queries.

When an ISP, for example, changes the location where an IP prefix is deployed, services which make use of geolocation information may begin to suffer degraded performance. This can lead to customer complaints, possibly to the ISP directly. Dissemination of correct geolocation data is complicated by the lack of any centralized means to coordinate and communicate geolocation information to all interested consumers of the data.

This document records a format whereby a network operator (an ISP, an enterprise, or any organization which deems the geolocation of its IP prefixes to be of concern) can publish a mapping of IP address prefixes to simplified geolocation information, colloquially termed a "geolocation feed". Interested parties can poll and parse these feeds to update or merge with other geolocation data sources and procedures.

Some technical organizations operating networks that move from one conference location to the next have already experimentally published

small geolocation feeds. At least one consumer (Google) has incorporated these ad hoc feeds into a geolocation data pipeline.

1.2. Requirements notation

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.

1.3. Implications of publication

This document describes both a format and a mechanism for publishing data, with the implication that the owner of the data wishes it to be public. Any privacy risk is bounded by the format, and feed publishers MAY omit any location field to further protect privacy (see Section 2.1 for details about which fields exactly may be omitted). Feed publishers assume the responsibility of determining which data should be made public.

This proposal does not incorporate a mechanism to communicate acceptable use policies for self-published data. Publication itself is inferred as a desire by the publisher for the data to be usefully consumed, similar to the publication of information like host names, cryptographic keys, and SPF records [RFC4408] in the DNS.

2. Self-published IP geolocation feeds

The format described here was developed to address the need of network operators to rapidly and usefully share geolocation information changes. Originally, there arose a specific case where regional operators found it desirable to publish location changes rather than wait for geolocation algorithms to "learn" about them. Later, technical conferences which frequently use the same network prefixes advertised from different conference locations experimented by publishing geolocation feeds, updated in advance of network location changes, in order to better serve conference attendees.

At its simplest, the mechanism consists of a network operator publishing a file (the "geolocation feed"), which contains several text entries, one per line. Each entry is keyed by a unique (within the feed) IP prefix (or single IP address) followed by a sequence of network locality attributes to be ascribed to the given prefix.

2.1. Specification

For operational simplicity, every feed should contain data about all IP addresses the provider wants to publish. Alternatives, like publishing only entries for IP addresses whose geolocation data has

changed or differ from current observed geolocation behavior "at large", are likely to be too operationally complex.

Feeds MUST use UTF-8 [RFC3629] character encoding. Text after a '#' character is treated as a comment only and ignored. Blank lines are similarly ignored.

Feeds MUST be in comma separated values format as described in [RFC4180]. Each feed entry is a text line of the form:

```
ip_prefix,country,region,city,postal_code
```

The IP prefix field is REQUIRED, all others are OPTIONAL (can be empty), though the requisite minimum number of commas SHOULD be present.

2.1.1. Geolocation feed individual entry fields

2.1.1.1. IP Prefix

REQUIRED. Each IP prefix field MUST be either a single IP address or an IP prefix in CIDR notation in conformance with section 3.1 [1] of [RFC4632] for IPv4 or section 2.3 [2] of [RFC4291] for IPv6.

Examples include "192.0.2.1" and "192.0.2.0/24" for IPv4 and "2001:db8::1" and "2001:db8::/32" for IPv6.

2.1.1.2. Country

OPTIONAL. The country field, if non-empty, MUST be a 2 letter ISO country code conforming to ISO 3166-1 alpha 2 [ISO.3166.1alpha2]. Parsers SHOULD treat this field case-insensitively.

Examples include "US" for the United States, "JP" for Japan, and "PL" for Poland.

2.1.1.3. Region

OPTIONAL. The region field, if non-empty, MUST be a ISO region code conforming to ISO 3166-2 [ISO.3166.2]. Parsers SHOULD treat this field case-insensitively.

Examples include "ID-RI" for the Riau province of Indonesia and "NG-RI" for the Rivers province in Nigeria.

2.1.1.4. City

OPTIONAL. The city field, if non-empty, SHOULD be free UTF-8 text, excluding the comma (',') character.

Examples include "Dublin", "New York", and "Sao Paulo" (specifically "S" followed by 0xc3, 0xa3, and "o Paulo").

2.1.1.5. Postal code

OPTIONAL. The postal code field, if non-empty, SHOULD be free UTF-8 text, excluding the comma (',') character. See Section 4 for some discussion of when this field must not be populated.

Examples include "106-6126" (in Minato ward, Tokyo, Japan).

2.1.2. Prefixes with no geolocation information

Feed publishers may indicate that some IP prefixes should not have any associated geolocation information. It may be that some prefixes under their administrative control are reserved, not yet allocated or deployed, or are in the process of being redeployed elsewhere and existing geolocation information can, from the perspective of the publisher, safely be discarded.

This special case can be indicated by explicitly leaving blank all fields which specify any degree of geolocation information. For example:

```
127.0.0.0/8,,,,
224.0.0.0/4,,,,
240.0.0.0/4,,,,
```

Historically, the user-assigned country identifier of "ZZ" had be used for this same purpose. This is not necessarily preferred, and no specific interpretation of any of the other user-assigned country codes is currently defined.

2.1.3. Additional parsing requirements

Feed entries missing required fields, or having a required field which fails to parse correctly MUST be discarded. It is RECOMMENDED that such entries also be logged for further administrative review.

While publishers SHOULD follow [RFC5952] style for IPv6 prefix fields, consumers MUST nevertheless accept all valid string representations.

Duplicate IP address or prefix entries MUST be considered an error, and consumer implementations SHOULD log the repeated entries for further administrative review. Publishers SHOULD take measures to ensure there is one and only one entry per IP address and prefix.

Feed entries with non-empty optional fields which fail to parse, either in part or in full, SHOULD be discarded. It is RECOMMENDED that they also be logged for further administrative review.

For compatibility with future additional fields a parser MUST ignore any fields beyond those it expects. The data from fields which are expected and which parse successfully MUST still be considered valid.

2.1.4. Looking up an IP address

Multiple entries which constitute nested prefixes are permitted. Consumers SHOULD consider the entry with the longest matching prefix (i.e. the "most specific") to be the best matching entry for a given IP address.

2.2. Examples

Example entries using different IP address formats and describing locations at country, region, city and postal code granularity level, respectively:

```
192.0.2.0/25,US,US-AL,,
192.0.2.5,US,US-AL,Alabaster,
192.0.2.128/25,PL,PL-MZ,,02-784
2001:db8::/32,PL,,,
2001:db8:cafe::/48,PL,PL-MZ,,02-784
```

The IETF network publishes geolocation information for the meeting prefixes, and generally just comment out the last meeting information and append the new meeting information. The [GEO_IETF] at the time of this writing contains:

```
# IETF 104, March 2019 - Prague, CZ.
# Note that Prague changed from CZ-PR to CZ-10 2016-11-15 - https://
www.iso.org/obp/ui/#iso:code:3166:CZ
130.129.0.0/16,CZ,CZ-10,Prague,
2001:df8::/32,CZ,CZ-10,Prague,
31.133.128.0/18,CZ,CZ-10,Prague,
31.130.224.0/20,CZ,CZ-10,Prague,
2001:67c:1230::/46,CZ,CZ-10,Prague,
2001:67c:370::/48,CZ,CZ-10,Prague,
```

Experimentally, RIPE has published geolocation information for their conference network prefixes, which change location in accordance with each new event. [GEO_RIPE_NCC] at the time of writing contains:

```
193.0.24.0/21,IS,IS-1,Reykjavik,
2001:67c:64::/48,IS,IS-1,Reykjavik,
```

Similarly, ICANN has published geolocation information for their portable conference network prefixes. [GEO_ICANN] at the time of writing contains:

```
199.91.192.0/21,ES,ES-CT,Barcelona
2620:f:8000::/48,ES,ES-CT,Barcelona
```

A longer example is the [GEO_Google] Google Corp Geofeed, which lists the geo-location information for Google coroprate offices.

Furthermore, it is worth noting that the geolocation data of SixXS users, already available at whois.sixxs.net, is now also accessible in the format described here (see [GEO_SIXXS]). This can be particularly useful where tunnel broker networks [RFC3053] are concerned as:

- the geolocation attributes of users with neighboring prefixes can be quite different and therefore not easily aggregated, and

- attempting to learn this data by statistical analysis can be complicated by the likely low number of samples for any given user, making satisfactory statistical confidence difficult to achieve.

2.3. Proposed extensions

Already some discussions have resulted in proposed extensions. While the purpose of this document is principally to record existing implementation details, it may be that there is a larger desire to publish other "network attributes" in a similar manner. One such network attribute, "delegation size", is not currently implemented but the state of the proposed extension is recorded here to demonstrate the flexibility required of parser implementations.

The following have been only informally discussed and are not in use at the time of writing.

2.3.1. Delegation size

OPTIONAL. A publisher may optionally communicate the average delegated prefix size for subnetworks within the IP prefix of this entry. For a network operator this can be used to help consumers distinguish IP prefixes among various use types such as residential prefixes, allocations to businesses, or data center customer allocations.

Non-empty strings MUST be of the form required for CIDR notation suffixes, i.e. "/" followed by the integer prefix length of the expected allocation to the subnetworks from within the entry's prefix. In the absence of data to the contrary, it is common to assume that leaf networks may be delegated a prefix ranging from /24 to /32 in IPv4 and /48 to /64 in IPv6. Default assumptions about delegation size are left to the consumer's implementation.

Examples for IPv6 include "/48", "/56", "/60", and "/64".

2.3.2. Alternate format

In order to more flexibly support future extensions, use of a more expressive feed format has been suggested. Use of JavaScript Object Notation (JSON, [RFC4627]), specifically, has been discussed. However, at the time of writing no such specification nor implementation exists.

3. Consuming self-published IP geolocation feeds

Consumers MAY treat published feed data as a hint only and MAY choose to prefer other sources of geolocation information for any given IP prefix. Regardless of a consumer's stance with respect to a given published feed, there are some points of note for sensibly and effectively consuming published feeds.

3.1. Feed integrity

The integrity of published information SHOULD be protected by securing the means of publication, for example by using HTTP over TLS [RFC2818]. Whenever possible, consumers SHOULD prefer retrieving geolocation feeds in a manner that guarantees integrity of the feed.

3.2. Verification of authority

Consumers of self-published IP geolocation feeds SHOULD perform some form of verification that the publisher is in fact authoritative for the addresses in the feed. The actual means of verification is likely dependent upon the way in which the feed is discovered. Ad

hoc shared URIs, for example, will likely require an ad hoc verification process. Future automated means of feed discovery SHOULD have an accompanying automated means of verification.

A consumer MUST only trust geolocation information for IP addresses or prefixes for which the publisher has been verified as administratively authoritative. All other geolocation feed entries MUST be ignored and SHOULD be logged for further administrative review.

3.3. Verification of accuracy

Errors and inaccuracies may occur at many levels, and publication and consumption of geolocation data are no exceptions. To the extent practical consumers SHOULD take steps to verify the accuracy of published locality. Verification methodology, resolution of discrepancies, and preference for alternative sources of data are left to the discretion of the feed consumer.

Consumers SHOULD decide on discrepancy thresholds and SHOULD flag for administrative review feed entries which exceed set thresholds.

3.4. Refreshing feed information

As a publisher can change geolocation data at any time and without notification consumers SHOULD implement mechanisms to periodically refresh local copies of feed data. In the absence of any other refresh timing information it is recommended that consumers SHOULD refresh feeds no less often than weekly.

For feeds available via HTTPS (or HTTP), the publisher MAY communicate refresh timing information by means of the standard HTTP expiration model (section 13.2 [3] of [RFC2616]). Specifically, publishers can include either an Expires header [4] or a Cache-Control header [5] specifying the max-age. Where practical, consumers SHOULD refresh feed information before the expiry time is reached.

4. Privacy Considerations

Publishers of geolocation feeds are advised to have fully considered any and all privacy implications of the disclosure of such information for the users of the described networks prior to publication. A thorough comprehension of the security considerations [6] of a chosen geolocation policy is highly recommended, including an understanding of some of the limitations of information obscurity [7] (see also [RFC6772]).

As noted in Section 2.1, each location field in an entry is optional, in order to support expressing only the level of specificity which the publisher has deemed acceptable. There is no requirement that the level of specificity be consistent across all entries within a feed. In particular, the Postal Code field (Section 2.1.1.5) can provide very specific geolocation, sometimes within a building. Such specific Postal Code values MUST NOT be published in geo feeds without the consent of the parties being located.

5. Relation to other work

While not originally done in conjunction with the [GEOPRIV] working group, Richard Barnes observed that this work is nevertheless consistent with that which the group has defined, both for address format and for privacy. The data elements in geolocation feeds are equivalent to the following XML structure (vis. [RFC5139]):

```
<civicAddress>
  <country>country</country>
  <A1>region</A1>
  <A2>city</A2>
  <PC>postal_code</PC>
</civicAddress>
```

Providing geolocation information to this granularity is equivalent to the following privacy policy (vis. the definition of the 'building' [8] level of disclosure):

```
<ruleset>
  <rule>
    <conditions/>
    <actions/>
    <transformations>
      <provide-location profile="civic-transformation">
        <provide-civic>building</provide-civic>
      </provide-location>
    </transformations>
  </rule>
</ruleset>
```

6. Security Considerations

As there is no true security in the obscurity of the location of any given IP address, self-publication of this data fundamentally opens no new attack vectors. For publishers, self-published data merely increases the ease with which such location data might be exploited.

For consumers, feed retrieval processes may receive input from potentially hostile sources (e.g. in the event of hijacked traffic). As such, proper input validation and defense measures MUST be taken.

Similarly, consumers who do not perform sufficient verification of published data bear the same risks as from other forms of geolocation configuration errors.

7. Finding self-published IP geolocation feeds

The issue of finding, and later verifying, geolocation feeds is not formally specified in this document. At this time, only ad hoc feed discovery and verification has a modicum of established practice (see below). Regardless, both the ad hoc mechanics and a few proposed but not yet implemented alternatives are discussed.

7.1. Ad hoc 'well known' URIs

To date, geolocation feeds have been shared informally in the form of HTTPS URIs exchanged in email threads. The two example URIs documented above describe networks that change locations periodically, the operators and operational practices of which are well known within their respective technical communities.

The contents of the feeds are verified by a similarly ad hoc process including:

- personal knowledge of the parties involved in the exchange, and
- comparison of feed-advertised prefixes with the BGP-advertised prefixes of Autonomous System Numbers known to be operated by the publishers.

Ad hoc mechanisms, while useful for early experimentation by producers and consumers, are unlikely to be adequate for long-term, widespread use by multiple parties. Future versions of any such self-published geolocation feed mechanism SHOULD address scalability concerns by defining a means for automated discovery and verification of operational authority of advertised prefixes.

7.2. Using public databases of network authority

One possibility for enabling automation would be publication of feed URIs as a well-known attribute in public databases of network authority, e.g. the WHOIS service ([RFC3912]) operated by RIRs. Verification may be performed if the same or similarly authoritative service provides the identical feed URI for queries for each CIDR prefix in the geolocation feed.

The burden of serving this data to all interested consumers, especially the load imposed by any verification process, is not yet known. The anticipation of additional operational burden on the public resource of record (the database of network authority) is however a noted concern.

7.3. Using 'reverse' DNS with NAPTR records

Another possibility for automating the location and verification of a geolocation feed is to incorporate feed URIs into the DNS, specifically the in-addr.arpa and ip6.arpa portions of the DNS hierarchy. A suitably formatted query for a NAPTR ([RFC3403]) record, or more specifically a U-NAPTR ([RFC4848]) record, could yield a transformation to a geolocation feed URI.

For example, assuming a purely theoretical service name of "x-geofeed", a 'reverse' DNS zone might contain a record of the form:

```
;;       order pref flags
IN NAPTR 200   10   "u"    "x-geofeed"        ( ; service
                                                 ; regexp
                           "!.*!https://example.com/ipgeo.csv!"
                           ""                    ; replacement
                           )
```

Attempts to locate the geolocation feed for a given IP address would begin by querying directly for a NAPTR record associated with the address's PTR-style name. For example, 192.0.2.4 and 2001:db8::6 would cause a NAPTR record request to be issued for "4.2.0.192.in-addr.arpa" and "6.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa", respectively.

If no such record exists one further NAPTR query for the fully qualified domain name of the SOA record in the authority section of the response to the previous query would be performed ("2.0.192.in-addr.arpa" and "d.0.1.0.0.2.ip6.arpa" in the examples above).

If one or more NAPTR records exist for the full PTR-style name but none of them are for the required service name (e.g. "x-geofeed"), then likely no SOA will be returned as a hint for subsequent queries. In this case implementations would need to first explicitly query for an SOA record for the full PTR-style name, and then query for a NAPTR record of the SOA in the response (assuming it differs from the previously queried name).

Any successfully located feed URIs could then be processed as outlined by this document.

Verification of the contents of a feed would proceed in essentially the same way. CIDR prefixes may be verified by constructing a query for any single address (at random) within the prefix and proceeding as above. While not strictly provably correct (in cases where a publisher has delegated some portion of the advertised prefix but not excluded it from its feed), it may nevertheless suffice for operational purposes, especially if a low-impact on-going verification of observed client IP addresses is implemented, to (eventually) catch any oversights.

This mode is untested and may prove impractical. However, the operational burden is more closely located with those wishing and willing to bear it, i.e. the publishers who would likely handle serving in-addr.arpa and ip6.arpa for the IP prefixes under their authority.

8. Acknowledgements

The authors would like to express their gratitude to reviewers and early implementers, including but not limited to Mikael Abrahamsson, Ray Bellis, John Bond, Alissa Cooper, Andras Erdei, Marco Hogewoning, Mike Joseph, Warren Kumari, Menno Schepers, Justyna Sidorska, Pim van Pelt, and Bjoern A. Zeeb. Richard L. Barnes in particular contributed substantial review, text, and advice.

9. References

9.1. Normative References

[ISO.3166.1alpha2]
International Organization for Standardization, "ISO 3166-1 decoding table", <http://www.iso.org/iso/home/standards/country_codes/iso-3166-1_decoding_table.htm>.

[ISO.3166.2]
International Organization for Standardization, "ISO 3166-2:2007", <http://www.iso.org/iso/home/standards/country_codes.htm#2012_iso3166-2>.

[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, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/info/rfc2616>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC4180] Shafranovich, Y., "Common Format and MIME Type for Comma-Separated Values (CSV) Files", RFC 4180, DOI 10.17487/RFC4180, October 2005, <https://www.rfc-editor.org/info/rfc4180>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4632] Fuller, V. and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan", BCP 122, RFC 4632, DOI 10.17487/RFC4632, August 2006, <https://www.rfc-editor.org/info/rfc4632>.

9.2. Informative References

[GEO_Google]
Google, LLC, "Google Corp Geofeed", <https://www.gstatic.com/geofeed/corp_external>.

[GEO_ICANN]
Internet Corporation For Assigned Names and Numbers, "ICANN Meeting Geolocation Data", <https://registration.icann.org/geo/google.csv>.

[GEO_IETF]
Kumari, A., "IETF Meeting Network Geolocation Data", <https://noc.ietf.org/geo/google.csv>.

[GEO_RIPE_NCC]
Schepers, M., "RIPE NCC Meeting Geolocation Data", <https://meetings.ripe.net/geo/google.csv>.

[GEO_SIXXS]
van Pelt, P., "SixXS Geolocation Data", <https://www.sixxs.net/export/google/>.

[GEOPRIV] Internet Engineering Task Force, "IETF geopriv Working Group", <http://datatracker.ietf.org/wg/geopriv/>.

[IPADDR_PY]
Shields, M. and P. Moody, "Python IP address manipulation library", <http://code.google.com/p/ipaddr-py/>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC3053] Durand, A., Fasano, P., Guardini, I., and D. Lento, "IPv6 Tunnel Broker", RFC 3053, DOI 10.17487/RFC3053, January 2001, <https://www.rfc-editor.org/info/rfc3053>.

[RFC3403] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, DOI 10.17487/RFC3403, October 2002, <https://www.rfc-editor.org/info/rfc3403>.

[RFC3912] Daigle, L., "WHOIS Protocol Specification", RFC 3912, DOI 10.17487/RFC3912, September 2004, <https://www.rfc-editor.org/info/rfc3912>.

[RFC4408] Wong, M. and W. Schlitt, "Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1", RFC 4408, DOI 10.17487/RFC4408, April 2006, <https://www.rfc-editor.org/info/rfc4408>.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <https://www.rfc-editor.org/info/rfc4627>.

[RFC4848] Daigle, L., "Domain-Based Application Service Location Using URIs and the Dynamic Delegation Discovery Service (DDDS)", RFC 4848, DOI 10.17487/RFC4848, April 2007, <https://www.rfc-editor.org/info/rfc4848>.

[RFC5139] Thomson, M. and J. Winterbottom, "Revised Civic Location Format for Presence Information Data Format Location Object (PIDF-LO)", RFC 5139, DOI 10.17487/RFC5139, February 2008, <https://www.rfc-editor.org/info/rfc5139>.

[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>.

[RFC6772] Schulzrinne, H., Ed., Tschofenig, H., Ed., Cuellar, J., Polk, J., Morris, J., and M. Thomson, "Geolocation Policy: A Document Format for Expressing Privacy Preferences for Location Information", RFC 6772, DOI 10.17487/RFC6772, January 2013, <https://www.rfc-editor.org/info/rfc6772>.

9.3. URIs

[1] http://tools.ietf.org/html/rfc4632#section-3.1

[2] http://tools.ietf.org/html/rfc4291#section-2.3

[3] http://tools.ietf.org/html/rfc2616#section-13.2

[4] http://tools.ietf.org/html/rfc2616#section-14.21

[5] http://tools.ietf.org/html/rfc2616#section-14.9

[6] http://tools.ietf.org/html/rfc6772#section-13

[7] http://tools.ietf.org/html/rfc6772#section-13.5

[8] http://tools.ietf.org/html/rfc6772#section-6.5.1

Appendix A. Sample Python validation code

Included here is a simple format validator in Python for self-published ipgeo feeds. This tool reads CSV data in the self-published ipgeo feed format from the standard input and performs basic validation. It is intended for use by feed publishers before launching a feed. Note that this validator does not verify the uniqueness of every IP prefix entry within the feed as a whole, but only verifies the syntax of each single line from within the feed. A complete validator MUST also ensure IP prefix uniqueness.

The main source file "ipgeo_feed_validator.py" follows. It requires use of the open source ipaddr Python library for IP address and CIDR parsing and validation [IPADDR_PY].

```
#!/usr/bin/python
#
# Copyright (c) 2012 IETF Trust and the persons identified as authors of
# the code. All rights reserved. Redistribution and use in source and
# binary forms, with or without modification, is permitted pursuant to,
# and subject to the license terms contained in, the Simplified BSD
# License set forth in Section 4.c of the IETF Trust's Legal Provisions
# Relating to IETF Documents (http://trustee.ietf.org/license-info).


"""Simple format validator for self-published ipgeo feeds.

This tool reads CSV data in the self-published ipgeo feed format from
the standard input and performs basic validation. It is intended for
use by feed publishers before launching a feed.
```

```
"""

import csv
import ipaddr
import re
import sys


class IPGeoFeedValidator(object):
  def __init__(self):
    self.prefixes = {}
    self.line_number = 0
    self.output_log = {}
    self.SetOutputStream(sys.stderr)

  def Validate(self, feed):
    """Check validity of an IPGeo feed.

    Args:
      feed: iterable with feed lines
    """

    for line in feed:
      self._ValidateLine(line)

  def SetOutputStream(self, logfile):
    """Controls where the output messages go do (STDERR by default).

    Use None to disable logging.

    Args:
      logfile: a file object (e.g., sys.stdout or sys.stderr) or None.
    """
    self.output_stream = logfile

  def CountErrors(self, severity):
    """How many ERRORs or WARNINGs were generated."""
    return len(self.output_log.get(severity, []))

  ############################################################
  def _ValidateLine(self, line):
    line = line.rstrip('\r\n')
    self.line_number += 1
    self.line = line.split('#')[0]
    self.is_correct_line = True

    if self._ShouldIgnoreLine(line):
      return
```

```
    fields = [field for field in csv.reader([line])][0]

    self._ValidateFields(fields)
    self._FlushOutputStream()

  def _ShouldIgnoreLine(self, line):
    line = line.strip()
    return len(line) == 0

  ############################################################
  def _ValidateFields(self, fields):
    assert(len(fields) > 0)

    is_correct = self._IsIPAddressOrPrefixCorrect(fields[0])

    if len(fields) > 1:
      if not self._IsCountryCode2Correct(fields[1]):
        is_correct = False

    if len(fields) > 2 and not self._IsRegionCodeCorrect(fields[2]):
      is_correct = False

    if len(fields) != 5:
      self._ReportWarning('5 fields were expected (got %d).'
                          % len(fields))

  ############################################################
  def _IsIPAddressOrPrefixCorrect(self, field):
    if '/' in field:
      return self._IsCIDRCorrect(field)
    return self._IsIPAddressCorrect(field)

  def _IsCIDRCorrect(self, cidr):
    try:
      ipprefix = ipaddr.IPNetwork(cidr)
      if ipprefix.network._ip != ipprefix._ip:
        self._ReportError('Incorrect IP Network.')
        return False
      if ipprefix.is_private:
        self._ReportError('IP Address must not be private.')
        return False
    except:
      self._ReportError('Incorrect IP Network.')
      return False
    return True

  def _IsIPAddressCorrect(self, ipaddress):
    try:
```

```
      ip = ipaddr.IPAddress(ipaddress)
    except:
      self._ReportError('Incorrect IP Address.')
      return False
    if ip.is_private:
      self._ReportError('IP Address must not be private.')
      return False
    return True

  ################################################################
  def _IsCountryCode2Correct(self, country_code_2):
    if len(country_code_2) == 0:
      return True
    if len(country_code_2) != 2 or not country_code_2.isalpha():
      self._ReportError(
          'Country code must be in the ISO 3166-1 alpha 2 format.')
      return False
    return True

  def _IsRegionCodeCorrect(self, region_code):
    if len(region_code) == 0:
      return True
    if '-' not in region_code:
      self._ReportError('Region code must be in the ISO 3166-2 format.')
      return False

    parts = region_code.split('-')
    if not self._IsCountryCode2Correct(parts[0]):
      return False
    return True

  ################################################################
  def _ReportError(self, message):
    self._ReportWithSeverity('ERROR', message)

  def _ReportWarning(self, message):
    self._ReportWithSeverity('WARNING', message)

  def _ReportWithSeverity(self, severity, message):
    self.is_correct_line = False
    output_line = '%s: %s\n' % (severity, message)

    if severity not in self.output_log:
      self.output_log[severity] = []
    self.output_log[severity].append(output_line)

    if self.output_stream is not None:
      self.output_stream.write(output_line)
```

```
  def _FlushOutputStream(self):
    if self.is_correct_line: return
    if self.output_stream is None: return

    self.output_stream.write('line %d: %s\n\n'
                             % (self.line_number, self.line))


################################################################
def main():
   feed_validator = IPGeoFeedValidator()
   feed_validator.Validate(sys.stdin)

   if feed_validator.CountErrors('ERROR'):
     sys.exit(1)

if __name__ == '__main__':
  main()
```

A unit test file, "ipgeo_feed_validator_test.py" is provided as well. It provides basic test coverage of the code above, though does not test correct handling of non-ASCII UTF-8 strings.

```
#!/usr/bin/python
#
# Copyright (c) 2012 IETF Trust and the persons identified as authors of
# the code.  All rights reserved.  Redistribution and use in source and
# binary forms, with or without modification, is permitted pursuant to,
# and subject to the license terms contained in, the Simplified BSD
# License set forth in Section 4.c of the IETF Trust's Legal Provisions
# Relating to IETF Documents (http://trustee.ietf.org/license-info).

import sys
from ipgeo_feed_validator import IPGeoFeedValidator

class IPGeoFeedValidatorTest(object):
  def __init__(self):
    self.validator = IPGeoFeedValidator()
    self.validator.SetOutputStream(None)
    self.successes = 0
    self.failures = 0

  def Run(self):
    self.TestFeedLine('# asdf', 0, 0)
    self.TestFeedLine('   ', 0, 0)
    self.TestFeedLine('', 0, 0)

    self.TestFeedLine('asdf', 1, 1)
```

```
    self.TestFeedLine('asdf,US,,,', 1, 0)
    self.TestFeedLine('aaaa::,US,,,', 0, 0)
    self.TestFeedLine('zzzz::,US', 1, 1)
    self.TestFeedLine(',US,,,', 1, 0)
    self.TestFeedLine('55.66.77', 1, 1)
    self.TestFeedLine('55.66.77.888', 1, 1)
    self.TestFeedLine('55.66.77.asdf', 1, 1)

    self.TestFeedLine('2001:db8:cafe::/48,PL,PL-MZ,,02-784', 0, 0)
    self.TestFeedLine('2001:db8:cafe::/48', 0, 1)

    self.TestFeedLine('55.66.77.88,PL', 0, 1)
    self.TestFeedLine('55.66.77.88,PL,,,', 0, 0)
    self.TestFeedLine('55.66.77.88,,,,', 0, 0)
    self.TestFeedLine('55.66.77.88,ZZ,,,', 0, 0)
    self.TestFeedLine('55.66.77.88,US,,,', 0, 0)
    self.TestFeedLine('55.66.77.88,USA,,,', 1, 0)
    self.TestFeedLine('55.66.77.88,99,,,', 1, 0)

    self.TestFeedLine('55.66.77.88,US,US-CA,,', 0, 0)
    self.TestFeedLine('55.66.77.88,US,USA-CA,,', 1, 0)
    self.TestFeedLine('55.66.77.88,USA,USA-CA,,', 2, 0)

    self.TestFeedLine('55.66.77.88,US,US-CA,Mountain View,', 0, 0)
    self.TestFeedLine('55.66.77.88,US,US-CA,Mountain View,94043', 0, 0)
    self.TestFeedLine('55.66.77.88,US,US-CA,Mountain View,94043,'
                      '1600 Ampthitheatre Parkway', 0, 1)

    self.TestFeedLine('55.66.77.0/24,US,,,', 0, 0)
    self.TestFeedLine('55.66.77.88/24,US,,,', 1, 0)
    self.TestFeedLine('55.66.77.88/32,US,,,', 0, 0)
    self.TestFeedLine('55.66.77/24,US,,,', 1, 0)
    self.TestFeedLine('55.66.77.0/35,US,,,', 1, 0)

    self.TestFeedLine('172.15.30.1,US,,,', 0, 0)
    self.TestFeedLine('172.28.30.1,US,,,', 1, 0)
    self.TestFeedLine('192.167.100.1,US,,,', 0, 0)
    self.TestFeedLine('192.168.100.1,US,,,', 1, 0)
    self.TestFeedLine('10.0.5.9,US,,,', 1, 0)
    self.TestFeedLine('10.0.5.0/24,US,,,', 1, 0)
    self.TestFeedLine('fc00::/48,PL,,,', 1, 0)
    self.TestFeedLine('fe00::/48,PL,,,', 0, 0)

    print '%d tests passed, %d failed' % (self.successes, self.failures)

  def IsOutputLogCorrectAtSeverity(self, severity, expected_msg_count):
    msg_count = self.validator.CountErrors(severity)
```

```
    if msg_count != expected_msg_count:
      print 'TEST FAILED: %s\nexpected %d %s[s], observed %d\n%s\n' % (
          self.validator.line, expected_msg_count, severity, msg_count,
          str(self.validator.output_log[severity]))
      return False
    return True

  def IsOutputLogCorrect(self, new_errors, new_warnings):
    retval = True

    if not self.IsOutputLogCorrectAtSeverity('ERROR', new_errors):
      retval = False
    if not self.IsOutputLogCorrectAtSeverity('WARNING', new_warnings):
      retval = False

    return retval

  def TestFeedLine(self, line, warning_count, error_count):
    self.validator.output_log['WARNING'] = []
    self.validator.output_log['ERROR'] = []
    self.validator._ValidateLine(line)

    if not self.IsOutputLogCorrect(warning_count, error_count):
      self.failures += 1
      return False

    self.successes += 1
    return True


if __name__ == '__main__':
  IPGeoFeedValidatorTest().Run()
```

Authors' Addresses

Erik Kline
Google Japan
Roppongi 6-10-1, 26th Floor
Minato, Tokyo 106-6126
Japan

Phone: +81 03 6384 9000
Email: ek@google.com

Krzysztof Duleba
Google Switzerland GmbH
Brandschenkestrasse 110
Zuerich 8002
Switzerland

Email: kduleba@google.com

Zoltan Szamonek
Google Switzerland GmbH
Brandschenkestrasse 110
Zuerich 8002
Switzerland

Email: zszami@google.com