Stream: Independent Submission
RFC: 8805
Category: Informational
Published: August 2020
ISSN: 2070-1721
Authors:
E. Kline, *Loon LLC*
K. Duleba, *Google*
Z. Szamonek, *Google Switzerland GmbH*
S. Moser, *Google Switzerland GmbH*
W. Kumari, *Google*

RFC 8805
A Format for Self-Published IP Geolocation Feeds

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.

This document describes a currently deployed format. At least one consumer (Google) has incorporated these feeds into a geolocation data pipeline, and a significant number of ISPs are using it to inform them where their prefixes should be geolocated.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8805.

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 that 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 that 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.

This document describes a currently deployed format. At least one consumer (Google) has incorporated these feeds into a geolocation data pipeline, and a significant number of ISPs are using it to inform them where their prefixes should be geolocated.

1.2. Requirements Notation

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

As this is an informational document about a data format and set of operational practices presently in use, requirements notation captures the design goals of the authors and implementors.

1.3. Assumptions about Publication

This document describes both a format and a mechanism for publishing data, with the assumption that the network operator to whom operational responsibility has been delegated for any published data wishes it to be public. Any privacy risk is bounded by the format, and feed publishers **MAY** omit prefixes or any location field associated with a given prefix 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 document 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 Sender Policy Framework (SPF) records [RFC7208] 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 that 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") that 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. Lines are delimited by a line break (CRLF) (as specified in [RFC4180]), and blank lines are ignored. Text from a '#' character to the end of the current line is treated as a comment only and is similarly ignored (note that this does not strictly follow [RFC4180], which has no support for comments).

Feed lines that are not comments **MUST** be formatted as comma-separated values (CSV), as described in [RFC4180]. Each feed entry is a text line of the form:

```
ip_prefix,alpha2code,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 Classless Inter-Domain Routing (CIDR) notation in conformance with Section 3.1 of [RFC4632] for IPv4 or Section 2.3 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. Alpha2code (Previously: 'country')

OPTIONAL: The alpha2code 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.

Earlier versions of this document called this field "country", and it may still be referred to as such in existing tools/interfaces.

Parsers **MAY** additionally support other 2-letter codes outside the ISO 3166-1 alpha 2 codes, such as the 2-letter codes from the "Exceptionally reserved codes" [ISO-GLOSSARY] set.

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 an 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, DEPRECATED: The postal code field, if non-empty, **SHOULD** be free UTF-8 text, excluding the comma (',') character. The use of this field is deprecated; consumers of feeds should be able to parse feeds containing these fields, but new feeds **SHOULD NOT** include this field due to the granularity of this information. See Section 4 for additional discussion.

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 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 that specify any degree of geolocation information. For example:

```
192.0.2.0/24,,,,
2001:db8:1::/48,,,,
2001:db8:2::/48,,,,
```

Historically, the user-assigned alpha2code identifier of "ZZ" has been used for this same purpose. This is not necessarily preferred, and no specific interpretation of any of the other user-assigned alpha2code codes is currently defined.

2.1.3. Additional Parsing Requirements

Feed entries that do not have an IP address or prefix field or have an IP address or prefix field that fails to parse correctly **MUST** be discarded.

While publishers **SHOULD** follow [RFC5952] 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.

Multiple entries that 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.

Feed entries with non-empty optional fields that 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 that are expected and that parse successfully **MUST** still be considered valid. Per Section 7, no extensions to this format are in use nor are any anticipated.

2.2. Examples

Example entries using different IP address formats and describing locations at alpha2code ("country code"), region, and city 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,,
2001:db8::/32,PL,,,
2001:db8:cafe::/48,PL,PL-MZ,,
```

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:

```
# IETF106 (Singapore) - November 2019 - Singapore, SG
130.129.0.0/16,SG,SG-01,Singapore,
2001:df8::/32,SG,SG-01,Singapore,
31.133.128.0/18,SG,SG-01,Singapore,
31.130.224.0/20,SG,SG-01,Singapore,
2001:67c:1230::/46,SG,SG-01,Singapore,
2001:67c:370::/48,SG,SG-01,Singapore,
```

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,NL,NL-ZH,Rotterdam,
2001:67c:64::/48,NL,NL-ZH,Rotterdam,
```

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,MA,MA-07,Marrakech
2620:f:8000::/48,MA,MA-07,Marrakech
```

A longer example is the [GEO_Google] Google Corp Geofeed, which lists the geolocation information for Google corporate offices.

At the time of writing, Google processes approximately 400 feeds comprising more than 750,000 IPv4 and IPv6 prefixes.

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 should only trust geolocation information for IP addresses or prefixes for which the publisher has been verified as administratively authoritative. All other geolocation feed entries should be ignored and 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 that 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 and no more often than is likely to cause issues to the publisher.

For feeds available via HTTPS (or HTTP), the publisher **MAY** communicate refresh timing information by means of the standard HTTP expiration model ([RFC7234]). Specifically, publishers can include either an Expires header (Section 5.3 of [RFC7234]) or a Cache-Control header (Section 5.2 of [RFC7234]) 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 (Section 13 of [RFC6772]) of a chosen geolocation policy is highly recommended, including an understanding of some of the limitations of information obscurity (Section 13.5 of [RFC6772]) (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 that 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 geofeeds without the express consent of the parties being located.

Operators who publish geolocation information are strongly encouraged to inform affected users/customers of this fact and of the potential privacy-related consequences and trade-offs.

5. Relation to Other Work

While not originally done in conjunction with the GEOPRIV Working Group [GEOPRIV], 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 ([RFC5139] [W3C.REC-xml-20081126]):

```
<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 (the definition of the 'building' Section 6.5.1 of [RFC6772] 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 may increase the ease with which such location data might be exploited (it can, for example, make easy the discovery of prefixes populated with customers as distinct from prefixes not generally in use).

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 (see the discussion in Section 3.1).

Similarly, consumers who do not perform sufficient verification of published data bear the same risks as from other forms of geolocation configuration errors (see the discussion in Sections 3.2 and 3.3).

Validation of a feed's contents includes verifying that the publisher is authoritative for the IP prefixes included in the feed. Failure to verify IP prefix authority would, for example, allow ISP Bob to make geolocation statements about IP space held by ISP Alice. At this time, only out-of-band verification methods are implemented (i.e., an ISP's feed may be verified against publicly available IP allocation data).

7. Planned Future Work

In order to more flexibly support future extensions, use of a more expressive feed format has been suggested. Use of JavaScript Object Notation (JSON) [RFC8259], specifically, has been discussed. However, at the time of writing, no such specification nor implementation exists. Nevertheless, work on extensions is deferred until a more suitable format has been selected.

The authors are planning on writing a document describing such a new format. This document describes a currently deployed and used format. Given the extremely limited extensibility of the present format no extensions to it are anticipated. Extensibility requirements are instead expected to be integral to the development of a new format.

8. 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); discussion of other mechanisms has been removed for clarity.

8.1. Ad Hoc 'Well-Known' URIs

To date, geolocation feeds have been shared informally in the form of HTTPS URIs exchanged in email threads. Three example URIs ([GEO_IETF], [GEO_RIPE_NCC], and [GEO_ICANN]) 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.

8.2. Other Mechanisms

Previous versions of this document referenced use of the WHOIS service [RFC3912] operated by Regional Internet Registries (RIRs), as well as possible DNS-based schemes to discover and validate geofeeds. To the authors' knowledge, support for such mechanisms has never been implemented, and this speculative text has been removed to avoid ambiguity.

9. IANA Considerations

This document has no IANA actions.

10. References

10.1. Normative References

[ISO.3166.1alpha2]

ISO, "ISO 3166-1 decoding table", <http://www.iso.org/iso/home/standards/country_codes/iso-3166-1_decoding_table.htm>.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[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>.

[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>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

10.2. Informative References

[GEOPRIV] IETF, "Geographic Location/Privacy (geopriv)", , <http://datatracker.ietf.org/wg/geopriv/>.

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

[GEO_ICANN] ICANN, "ICANN Meeting Geolocation Data", <https://meeting-services.icann.org/geo/google.csv>.

[GEO_IETF] Kumari, W., "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>.

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

[ISO-GLOSSARY] ISO, "Glossary for ISO 3166", , <https://www.iso.org/glossary-for-iso-3166.html>.

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

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

[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>.

[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>.

[RFC7208] Kitterman, S., "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1", RFC 7208, DOI 10.17487/RFC7208, April 2014, <https://www.rfc-editor.org/info/rfc7208>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

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].

```
<CODE BEGINS>
#!/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 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()
    if line.startswith('#'):
      return True
    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._IsAlpha2CodeCorrect(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 _IsAlpha2CodeCorrect(self, alpha2code):
    if len(alpha2code) == 0:
      return True
    if len(alpha2code) != 2 or not alpha2code.isalpha():
      self._ReportError(
          'Alpha 2 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 ISO 3166-2 format.')
      return False

    parts = region_code.split('-')
    if not self._IsAlpha2CodeCorrect(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()

<CODE ENDS>
```

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.

```
<CODE BEGINS>
#!/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()

<CODE ENDS>
```

Acknowledgements

The authors would like to express their gratitude to reviewers and early implementors, including but not limited to Mikael Abrahamsson, Andrew Alston, Ray Bellis, John Bond, Alissa Cooper, Andras Erdei, Stephen Farrell, Marco Hogewoning, Mike Joseph, Maciej Kuzniar, George Michaelson, Menno Schepers, Justyna Sidorska, Pim van Pelt, and Bjoern A. Zeeb.

In particular, Richard L. Barnes and Andy Newton contributed substantial review, text, and advice.

Authors' Addresses

Erik Kline
Loon LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: ek@loon.com

Krzysztof Duleba
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: kduleba@google.com

Zoltan Szamonek
Google Switzerland GmbH
Brandschenkestrasse 110
CH-8002 Zürich
Switzerland
Email: zszami@google.com

Stefan Moser
Google Switzerland GmbH
Brandschenkestrasse 110
CH-8002 Zürich
Switzerland
Email: smoser@google.com

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: warren@kumari.net