Operations and Management Area Working Group
Internet-Draft
Updates: RFC8805 (if approved)
Intended status: Informational
Expires: 18 November 2026

W. Kumari
Google LLC

17 May 2026

A JSON Format for Self-Published IP Geolocation Feeds
draft-wkumari-opsawg-json-geofeed-format-00

Abstract

This document defines a JavaScript Object Notation (JSON) format for self-published IP geolocation feeds. It updates RFC 8805 by transitioning from the current comma-separated values (CSV) format to a more expressive JSON format, addressing the need for operational extensibility.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://wkumari.github.io/draft-wkumari-opsawg-json-geofeed-format/draft-wkumari-opsawg-json-geofeed-format.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-wkumari-opsawg-json-geofeed-format/.

Discussion of this document takes place on the Operations and Management Area Working Group Working Group mailing list (mailto:opsawg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/opsawg/. Subscribe at https://www.ietf.org/mailman/listinfo/opsawg/.

Source for this draft and an issue tracker can be found at https://github.com/wkumari/draft-wkumari-opsawg-json-geofeed-format.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC8805] Section 2.1 defines a CSV-based format for network operators to publish a mapping of IP address prefixes to simplified geolocation information. While widely deployed, the authors of [RFC8805] acknowledged in Section 7that the CSV format has "extremely limited extensibility" and that future work should involve the development of a more expressive format, specifically suggesting JSON (see[RFC8805], Section 7. "Planned Future Work").

Furthermore, [IAB-IP-GEO] identified several critical gaps in the existing geofeed ecosystem, including:

* The CSV format cannot adequately express varying levels of confidence in a location mapping ([IAB-IP-GEO], Section 4.2), nor can it map a prefix to multiple locations ([IAB-IP-GEO], Section 5.2) for example if the prefix is used for Anycast.

* The current format lacks the ontology to clarify whether the geolocation mapping refers to the physical location of the user, the location of network infrastructure, a network egress point, or a regulatory jurisdiction ([IAB-IP-GEO], Section 3.3).

Note that [IAB-IP-GEO] also identified other, more architectural issues, including that physical location does not necessarily correspond to network topologies, the lack of user consent mechanisms, the potential for privacy violations, the need for different levels of precision (e.g. "get me to a close datacenter" vs "the ambulance needs my physical address", etc).

However, these issues, while very important, are orthogonal to the data format and should be addressed through other mechanisms. This document simply tries to improve the data format to better support the ecosystem as it currently exists, while providing a framework for future extensions.

Readers are strongly encouraged to read [IAB-IP-GEO], the accompanying papers from the IAB Workshop on IP Address Geolocation for a deeper understanding of these architectural issues and the broader context of geolocation in the modern internet. [KLINE-GEO] and [SZAMONEK-GEO] are particularly recommended to understand the architecture, use-case and privacy considerations in the original design.

This document defines defines a new JSON-based format to address the extensibility limitations of the current CSV format, while maintaining compatibility with the core fields defined in [RFC8805]. It also introduces new fields to address some of the gaps identified in [IAB-IP-GEO].

2. Conventions and Definitions

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.

3. JSON Geofeed Format

3.1. Metadata section

A compliant geofeed MUST include a new section comprised of a JSON object including certain fields which improve the operational usefulness of the information within.

REQUIRED fields in this section include:

* *last_updated*: The time and date the feed was last generated, formatted as an ISO 8601 date-time.

* *contact*: An email addresss or URL (e.g., for a web form) for outreach about the geofeed for operational or other issues.

* *update_frequency*: The time, expressed either in a number of seconds or as an ISO 8601 duration. E.g., 86400 (seconds) or P1D (ISO 8601) each represent a daily interval.

OPTIONAL fields in this section include:

* *source*: The type of entity generating the geofeed information. Values include ISP, CDN, geo_provider, registry.

* *applicability_statement*: A text field describing how the data is intended to be used.

3.2. Body section

The JSON format for geolocation feeds builds upon the fields specified in RFC 8805: IP prefix, alpha2code, region, city.

Note that, as [RFC8805] deprecated the use of the postal code field ([RFC8805], Section 2.1.1.5 - "Postal Code"), the JSON format does not support it.

A JSON geofeed MUST be an array of JSON objects.

Each object MUST contain the following keys:

* *ip_prefix*: same semantics as defined in [RFC8805]

* *alpha2code*: same semantics as defined in [RFC8805]

* *region*: same semantics as defined in [RFC8805]

* *city*: same semantics as defined in [RFC8805]

In addition, each object MUST contain:

* *last_updated*: A string indicating the timestamp of the last update to that record, formatted as an ISO 8601 date-time. This field is critical for consumers to assess the freshness of the data, given the dynamic nature of IP address allocations and network changes.

Objects MAY contain the following optional keys:

* *location_type* (optional): A string indicating the nature of the location.

* Valid values include infrastructure, network_egress, organization and jurisdiction.

* *confidence* (optional): A string expressing the certainty level of the mapping ([IAB-IP-GEO], Section 5.2). Valid values include high, medium, and low.

Example JSON Geofeed Entry:

```
[
  {
    "ip_prefix": "192.0.2.0/24",
    "alpha2code": "US",
    "region": "US-AL",
    "city": "Alabaster",
    "location_type": "infrastructure",
    "confidence": "high",
    "last_updated": "2024-06-01T12:00:00Z"
  },
  {
    "ip_prefix": "198.51.100.0/24",
    "alpha2code": "CZ",
    "region": "CZ-PR",
    "city": "Praha",
    "location_type": "network_egress",
    "confidence": "medium",
    "last_updated": "2017-07-01T12:00:00Z"
  }
]
```

The location_type field allows publishers to clarify the context of the geolocation mapping, while the confidence field provides a mechanism to express the reliability of the data. The last_updated field helps consumers assess the freshness of the information, which is crucial given the dynamic nature of IP address allocations and network changes.

{*Editor note*: Multiple people have suggested new fields that should be added to the format, but we need to be careful about scope creep. These are only some of the proposed fields, but any others such as source, accuracy_radius and access_technology have also been discussed. For example, access_technology would be used to indicate the type of network access, such as wifi, cellular, or ethernet, and / or speed, jitter, etc.

I have intentionally kept the initial list the same as RFC8805, but added last_updated as it was obviously needed, and location_type and confidence as examples.

There will need to be careful discussion of other potential fields before they are added to the format, and it is possible that there will need to be a dedicated venue for such discussions. In addition, we might consider creating a registry for these fields to ensure consistency and interoperability, and a mechanism to provide "private" extensions.}

4. Security Considerations

As noted in RFC 8805, self-publication of location data opens no new attack vectors, but consumers must validate inputs from potentially hostile sources.

The JSON format allows for greater specificity, which increases the necessity for publishers to verify they have operational authority over the advertised prefixes and to ensure the accuracy of the geolocation data. Consumers should also be aware of the potential for stale data and consider the last_updated field when making decisions based on geofeed information.

5. IANA Considerations

This document has no IANA actions.

6. References

6.1. Normative References

[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/rfc/rfc2119>.

[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/rfc/rfc8174>.

[RFC8805] Kline, E., Duleba, K., Szamonek, Z., Moser, S., and W. Kumari, "A Format for Self-Published IP Geolocation Feeds", RFC 8805, DOI 10.17487/RFC8805, August 2020, <https://www.rfc-editor.org/rfc/rfc8805>.

6.2. Informative References

[IAB-IP-GEO] Iyengar, J., Livingood, J., and T. Pauly, "Report from the IAB Workshop on IP Address Geolocation", 8 April 2026.

[KLINE-GEO] Kline, E., "Anecdotal History of RFC 8805", November 2025, <https://www.ietf.org/slides/slides-ipgeows-paper-anecdotal-history-of-rfc-00.pdf>.

[SZAMONEK-GEO] Szamonek, Z., "The Need for an Alternative to IP-Based Geolocation", November 2025, <https://www.ietf.org/slides/slides-ipgeows-paper-the-need-for-an-alternative-to-ip-based-geolocation-00.pdf>.

Acknowledgments

The authors would like to thank the participants of the IAB Workshop on IP Address Geolocation for their valuable insights and feedback, which have greatly informed the development of this document.

In particular, the authors would like to thank the following individuals for their contributions to the discussions that led to the development of this document: Nimrod Levy, Jari Arkko, Brian Trammell, Erik Kline, Erik Nygren, Geoff Huston, Jari Arkko, Jason Livingood, Joe Abley, Joe Clarke, Joel Jaeggli, Jana Iyengar, Lee Howard, Robert Kisteleki, Tommy Pauly and Zoltan Szamonek.

The authors would especially like to thank Tony Tauber for submitting useful pull requests.

Appendix A: Example Conversion Script

The following Python program (convert.py) demonstrates how to convert an RFC 8805 format CSV geofeed into the JSON format defined in this document. This script reads from standard input and outputs the JSON to standard output.

Note that this is a simple example script and does not include error handling, validation, or support for all potential fields. It is intended for illustrative purposes only.

--- CODE BEGINS ---

```
import sys
import csv
import json
from datetime import datetime, timezone


def process_csv(input_stream):
    """
    This parses RFC8805 format CSV and returns a JSON formatted string.
    """
    # Get current time in UTC, formatted as ISO 8601
    current_time = datetime.now(timezone.utc).strftime("%Y-%m-%dT%H:%M:%SZ")

    json_records = []

    reader = csv.DictReader(
        input_stream, fieldnames=["ip_prefix", "alpha2code", "region", "city"]
    )

    for row in reader:
        # Skip comment lines
        if row["ip_prefix"].startswith("#"):
            continue

        # Construct the dictionary using the keys from the CSV header
        record = {
            "ip_prefix": row.get("ip_prefix", ""),
            "alpha2code": row.get("alpha2code", ""),
            "region": row.get("region", ""),
            "city": row.get("city", ""),
            "last_updated": current_time,
        }

        json_records.append(record)

    return json.dumps(json_records, indent=2)


if __name__ == "__main__":
    output = process_csv(sys.stdin)
    print(output)
```

--- CODE ENDS ---

The following Python program (test_convert.py) includes unit tests for the conversion script, demonstrating how to test the conversion of CSV input into the expected JSON output.

--- CODE BEGINS ---

```
import unittest
import io
import json
from convert import process_csv


class TestCSVParser(unittest.TestCase):

    # Examples from the RFC8805 document
    def test_standard_ipv4_parsing(self):
        csv_data = "192.0.2.5,US,US-AL,Alabaster,\n"
        # Simulate an input stream
        stream = io.StringIO(csv_data)

        result_json = process_csv(stream)
        data = json.loads(result_json)

        # Verify we got exactly one record
        self.assertEqual(len(data), 1)

        # Verify the dynamically parsed fields
        self.assertEqual(data[0]["ip_prefix"], "192.0.2.5")
        self.assertEqual(data[0]["alpha2code"], "US")
        self.assertEqual(data[0]["region"], "US-AL")
        self.assertEqual(data[0]["city"], "Alabaster")

        # Verify static/injected fields
        self.assertTrue("T" in data[0]["last_updated"])  # Quick check for date format

    def test_ipv6_and_empty_fields(self):
        csv_data = "2001:db8::1,US,,,\n"
        stream = io.StringIO(csv_data)

        result_json = process_csv(stream)
        data = json.loads(result_json)

        self.assertEqual(data[0]["ip_prefix"], "2001:db8::1")
        self.assertEqual(data[0]["alpha2code"], "US")
        # Ensure missing fields evaluate to empty strings
        self.assertEqual(data[0]["region"], "")
        self.assertEqual(data[0]["city"], "")

    def test_ignore_comment_records(self):
        csv_data = (
            "# IETF106 (Singapore) - November 2019 - Singapore, SG\n"
            "130.129.0.0/16,SG,SG-01,Singapore,"
```

```
        )
        stream = io.StringIO(csv_data)

        result_json = process_csv(stream)
        data = json.loads(result_json)

        # Data should contain only the non-comment record
        self.assertEqual(len(data), 1)

        # And the fields should be correctly parsed. 2 tests for the price of one!
        self.assertEqual(data[0]["ip_prefix"], "130.129.0.0/16")
        self.assertEqual(data[0]["alpha2code"], "SG")
        self.assertEqual(data[0]["region"], "SG-01")
        self.assertEqual(data[0]["city"], "Singapore")

    def test_empty_input(self):
        stream = io.StringIO("")
        result_json = process_csv(stream)
        data = json.loads(result_json)

        self.assertEqual(data, [])


if __name__ == "__main__":
    unittest.main()
```

--- CODE ENDS ---

Author's Address

Warren Kumari
Google LLC
Email: warren@kumari.net