DISPATCH
Internet-Draft
Intended status: Informational
Expires: 22 February 2026

M. Kucherawy, Ed.

W. Kumari
R. Sloan
Google

21 August 2025

# Media Type Registration for Protocol Buffers
draft-ietf-dispatch-mime-protobuf-03

## Abstract

This document registers media types for Protocol Buffers, a common extensible mechanism for serializing structured data.

## 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://github.com/wkumari/draft-murray-dispatch-mime-protobuf. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-dispatch-mime-protobuf/.

Discussion of this document takes place on the DISPATCH Working Group mailing list (mailto:dispatch@ietf.org), which is archived at https://www.ietf.org/mailman/listinfo/dispatch. Subscribe at https://www.ietf.org/mailman/listinfo/dispatch/.

Source for this draft and an issue tracker can be found at https://github.com//wkumari/draft-murray-dispatch-mime-protobuf.

## 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 22 February 2026.

Copyright Notice

Copyright (c) 2025 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents



## 1. Introduction

Protocol Buffers ("protobufs") were introduced in 2008 as a free, open source, platform-independent mechanism for transport and storage of structured data: their use has become increasingly common and Protobuf implementations exist in many languages (C++, C#, Dart, Go, Java, Kotlin, Objective-C, Python, JavaScript, Ruby, Swift, and perhaps others). See [Protobuf] for more information.

Protobuf consists of an interface definition language ("IDL"), wire encoding formats, and language-specific implementations (typically involving a generated API) so that clients and servers can be easily deployed using a common schema. Protobuf supports multiple wire

formats for interchange: [Binary], which is optimized for wire efficiency, and [ProtoJSON], which maps the Protobuf schema onto a JSON structure.

Serialized objects are occasionally transported within media that make use of media types (see [RFC2045] et seq) to identify payloads. Accordingly, current and historical media types used for this purpose would benefit from registration. This document requests those registrations of IANA.

## 2. Key Words

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. Payload Description

These media types are used in the transport of serialized objects only. The IDL and object definitions, if transported, would be used with any appropriate text media type. In the three figures below, only the third of these would ever be used with these media types (a JSON example is depicted).

An example use of the IDL to specify a "Person" object:

```
edition = "2023";

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;
}
```

An example of python code that uses code generated from the IDL definition above to create an instance of a "Person" object:

```
person = Person()
person.id = 1234
person.name = "John Doe"
person.email = "jdoe@example.com"
```

An example of the above instance expressed in JSON:

```
{
  "name": "John Doe",
  "id": 1234,
  "email": "jdoe@example.com"
}
```

## 4. Encoding Considerations

Protobuf supports only the [Binary] and [ProtoJSON] formats for interchange, both of which are platform-independent. For binary forms that need to transit non-binary transports, a base64 Content-Transfer-Encoding (xref to [RFC4648]) is recommended.

The media type includes an optional "encoding" parameter indicating which encoding format is to be used with that particular payload. This is included for future extensibility. Valid values for this parameter are "binary" and "json", and other values MUST be treated as an error. See Section 7 for the defaults for each of the two registered media types. Using "binary" for the JSON type or "json" for the binary type MUST be treated as an error.

## 5. Versions

[Proto2] was the first public version of the Protobuf schema language, and [Proto3] and [Edition2023] came later, with the last of these being current at the time of writing. Future editions of the IDL are expected.

These versions refer to evolutions of the schema of the IDL, not the wire format. Accordingly, a serialized object generated by any of these is compatible with any other. The media type registrations in Section 7 include support for versioning of the wire format, should it ever change, but do not refer to the IDL, which can evolve independently.

Note that there may be semantic changes implicit in the IDL version which can affect the interpretation of otherwise compatible bits on the wire. For example, in Proto2, unknown values of an enumeration were interpreted as invalid, whereas in Proto3 they are retained.

Clients MUST reject payloads with an unsupported version number.

## 6. Security Considerations

The payload for these media types contain no directly executable code. While it is common for a protobuf definition to be used as input to a code generator which then produces something executable, but that applies to the schema language, not serializations.

Protobuf provides no security, privacy, integrity, or compression services: clients or servers for which this is a concern should avail themselves of solutions that provide such capabilities (e.g. [RFC8446]). Implementations should be careful when processing Protobuf like any binary format: a malformed request to a protobuf server could be crafted to, for example, allocate a very large amount of memory, potentially impacting other operations on that server.

Protobuf supports embedded content in string or bytes fields: in both cases, applications should ensure that the format of the content is precisely as expected. Note that UTF-8 validation of string fields is optional (see [ProtoFeatures]) and a manual well-formedness check may be necessary. Further, handling Unicode text generally can be quite complex with problems discussed, for example, in [UniChars] and [RFC8264]; so it is best to rely on well-supported internationalization libraries whenever possible.

In order to safely use Protobuf serializations on the web, it is important to ensure that they are not interpreted as another document type, such as JavaScript: we recommend base64-encoding binary Protobuf responses whenever possible to prevent parsing as active content. Servers should generally follow the advice of [RFC9205] to prevent content sniffing for all binary formats.

Further, when using JSON serializations it is important that it is clear to browsers that the content is pure JSON, so that they can inhibit Cross-Site Script Inclusion or side-channel attacks using techniques such as Cross-Origin Read Blocking ([CORB]). Per [RFC6839], pure JSON content is indicated by a +json subtype suffix (see also [MIMESNIFF]); so when serializing Protobuf content to JSON, users MUST use the application/protobuf+json media type. When using JSON, charset can prevent certain encoding confusion attacks so users should specify it for all JSON encodings.

In the [Any] type there is technically a link, which was intended to be dereferenced to obtain schemas for a given type; however this is not supported by widely used Protobuf implementations.

## 7. IANA Considerations

As per the process defined in [RFC6838], this document requests the registration of application/protobuf and application/protobuf+json as media types for Protobuf, and the notation of application/x-protobuf, application/x-protobuffer, and application/x-protobuf+json as deprecated aliases:

### 7.1. Registration for the "application/protobuf" Media Type

Type name: application

Subtype name: protobuf

Required parameters: none

Optional parameters:

* encoding, which indicates the type of Protobuf encoding and is "binary" by default for application/protobuf, indicating the [Binary] format. At the time of writing, no other encoding can be used for application/protobuf so this parameter is for extensibility.

* version, which indicates the version of the wire encoding specification (not the schema language), with default 1. At the time of writing, no protobuf wire encodings are versioned so this parameter is for extensibility. Unversioned wire encodings should be treated as having version 1. Protobuf implementations should accept all versions of wire encodings defined at the time of implementation.

Encoding considerations: binary

Security considerations: see Section 6

Interoperability considerations: The Protobuf specification includes versioning provisions to ensure backward compatibility when encountering payloads with unknown properties.

Published specification: [Protobuf]

Applications that use this media type: Any application with a need to exchange or store structured objects across platforms or implementations.

Fragment identifier considerations: None.

Additional information:

* Deprecated alias names for this type: application/x-protobuf, application/x-protobuffer

* Magic number(s):

* File extension(s):

* Macintosh file type code(s):

Person & email address to contact for further information: Protobuf <protobuf-team@google.com>

Intended usage: COMMON

Restrictions on usage: None

Author: Rob Sloan <rmsj@google.com>

Change controller: Protobuf <protobuf-team@google.com>

Provisional registration? (standards tree only): No

### 7.2. Registration for "application/protobuf+json" Media Type

Type name: application

Subtype name: protobuf+json

Required parameters: charset, which must be set to utf-8 (case-insensitive)

Optional parameters:

* encoding, which indicates the type of Protobuf encoding and is json by default for application/protobuf+json, indicating the [ProtoJSON] format. At the time of writing, no other encoding can be used for application/protobuf+json so this parameter is for extensibility.

* version, which indicates the version of the wire encoding specification (not the schema language), with default 1. At the time of writing, no protobuf wire encodings are versioned so this parameter is for extensibility. Unversioned wire encodings should be treated as having version 1. Protobuf implementations should accept all versions of wire encodings defined at the time of implementation.

Encoding considerations: Same as encoding considerations of application/json as specified in [RFC8259], Section 11.

Security considerations: see Section 6

Interoperability considerations: The Protobuf specification includes versioning provisions to ensure backward compatibility when encountering payloads with unknown properties.

Published specification: [Protobuf]

Applications that use this media type: Any application with a need to exchange or store structured objects across platforms or implementations.

Fragment identifier considerations: None.

Additional information:

Deprecated alias names for this type: x-protobuf+json
Magic number(s):
File extension(s):
Macintosh file type code(s):

Person & email address to contact for further information: Protobuf <protobuf-team@google.com>

Intended usage: COMMON

Restrictions on usage: None

Author: Rob Sloan <rmsj@google.com>

Change controller: Protobuf <protobuf-team@google.com>

Provisional registration? (standards tree only): No

## 8. Contact

Please contact protobuf-team@google.com for requests to adjust this specification. Issues may be raised at https://github.com/protocolbuffers/protobuf.

## 9. References

### 9.1. Normative References

[Protobuf] "Protocol Buffers", n.d., <https://protobuf.dev/>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/rfc/rfc2045>.

[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>.

[RFC4648]  Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC6838]  Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[RFC6839]  Hansen, T. and A. Melnikov, "Additional Media Type Structured Syntax Suffixes", RFC 6839, DOI 10.17487/RFC6839, January 2013, <https://www.rfc-editor.org/rfc/rfc6839>.

[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>.

[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/rfc/rfc8259>.

## 9.2. Informative References

[Any]  Protobuf, "any.proto Schema Definition", n.d., <https://github.com/protocolbuffers/protobuf/blob/main/src/google/protobuf/any.proto>.

[Binary]  Protobuf, "Protobuf Binary Wire Encoding Spec", n.d., <https://protobuf.dev/programming-guides/encoding>.

[CORB]  Chromium, "Cross-Origin Read Blocking for Web Developers", n.d., <https://www.chromium.org/Home/chromium-security/corb-for-developers>.

[Edition2023]  Protobuf, "Proto Edition 2023 Schema Language Specification", n.d., <https://protobuf.dev/reference/protobuf/edition-2023-spec>.

[MIMESNIFF]
WHATWG, "MIME Sniffing: Living Standard", n.d.,
<https://mimesniff.spec.whatwg.org/#mime-type-groups>.

[Proto2] Protobuf, "Proto2 Schema Language Specification", n.d.,
<https://protobuf.dev/reference/protobuf/proto2-spec>.

[Proto3] Protobuf, "Proto3 Schema Language Specification", n.d.,
<https://protobuf.dev/reference/protobuf/proto3-spec>.

[ProtoFeatures]
Protobuf, "Protobuf Feature Settings for Editions", n.d.,
<https://protobuf.dev/editions/features/>.

[ProtoJSON]
Protobuf, "Protobuf JSON Wire Encoding Spec", n.d.,
<https://protobuf.dev/programming-guides/json>.

[RFC8264] Saint-Andre, P. and M. Blanchet, "PRECIS Framework:
Preparation, Enforcement, and Comparison of
Internationalized Strings in Application Protocols",
RFC 8264, DOI 10.17487/RFC8264, October 2017,
<https://www.rfc-editor.org/rfc/rfc8264>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol
Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018,
<https://www.rfc-editor.org/rfc/rfc8446>.

[RFC9205] Nottingham, M., "Building Protocols with HTTP", BCP 56,
RFC 9205, DOI 10.17487/RFC9205, June 2022,
<https://www.rfc-editor.org/rfc/rfc9205>.

[UniChars] IETF, "Unicode Character Repertoire Subsets", n.d.,
<https://datatracker.ietf.org/doc/draft-bray-unichars/>.

## Acknowledgments

Orie Steele provided valuable feedback to this work.

## Authors' Addresses

Murray S. Kucherawy (editor)
Email: superuser@gmail.com

Warren Kumari
Google
Email: warren@kumari.net

Rob Sloan
Google
Email: rmsj@google.com