Stream:	Internet Engineering Task Force (IETF)		
RFC:	9996		
Category:	Informational		
Published:	July 2026		
ISSN:	2070-1721		
Authors:	M. Kucherawy, Ed.	W. Kumari	R. Sloan
		Google	*Google*

RFC 9996
Media Types for Protocol Buffers

Abstract

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

Status of This Memo

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

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are 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/rfc9996.

Table of Contents

1. Introduction

Protocol Buffers (also called "Protobuf") were introduced in 2008 as a free, open-source, platform-independent mechanism for transport and storage of structured data. The use of Protobuf 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 two wire formats for interchange: the format in [Binary] (which is optimized for wire efficiency) and the format in [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]) to identify payloads. Accordingly, current and historical media types used for this purpose would benefit from registration. IANA has registered two Protobuf media types; see Section 7.

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

The media types defined in this document 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 examples below, only the third (depicted in JSON) would ever be used with these media types.

1. An example of using the IDL to specify a "Person" object:

```
edition = "2023";

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

2. 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"
```

3. An example of the above instance expressed in JSON:

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

4. Encoding Considerations

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

Both of the media types defined in this document include 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; [Proto3], [Edition2023], and [Edition2024] came later, with [Edition2024] 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 contains 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, 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; for example, a malformed request to a Protobuf server could be crafted to 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 (due to the problems discussed in [UniChars] and [RFC8264], for example), 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]); thus, 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 type described in [Any], there is technically a link that 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], IANA has registered `application/protobuf` and `application/protobuf+json` in the "Media Types" registry. The following are deprecated aliases: `application/x-protobuf`, `application/x-protobuffer`, and `application/x-protobuf+json`.

7.1. Media Type: `application/protobuf`

Type name: application

Subtype name: protobuf

Required parameters: N/A

Optional parameters:

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

`version`
: Indicates the version of the wire encoding specification (not the schema language), with a default of `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`.

Encoding considerations: binary

Security considerations: See Section 6 of RFC 9996.

Interoperability considerations: The Protobuf specification [Protobuf] 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: N/A

Additional information:

Deprecated alias names for this type: `application/x-protobuf`, `application/x-protobuffer`
Magic number(s): N/A
File extension(s): N/A
Macintosh file type code(s): N/A

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: Rob Sloan <rmsj@google.com>, Protobuf Team <protobuf-team@google.com>

Change controller: IETF

7.2. Media Type: `application/protobuf+json`

Type name: application

Subtype name: protobuf+json

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

Optional parameters:

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

`version`
: Indicates the version of the wire encoding specification (not the schema language), with a default of `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`.

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

Security considerations: See Section 6 of RFC 9996.

Interoperability considerations: The Protobuf specification [Protobuf] 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: N/A

Additional information:

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

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: Rob Sloan <rmsj@google.com>, Protobuf Team <protobuf-team@google.com>

Change controller: IETF

8. References

8.1. Normative References

[Protobuf] Google, LLC, "Protocol Buffers", <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/info/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/info/rfc2119>.

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

[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/info/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/info/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/info/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/info/rfc8259>.

8.2. Informative References

[Any] "any.proto Schema Definition", commit 97921a5, 12 February 2026, <https://github.com/protocolbuffers/protobuf/blob/main/src/google/protobuf/any.proto>.

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

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

[Edition2023] Google, LLC, "Protocol Buffers Edition 2023 Language Specification", <https://protobuf.dev/reference/protobuf/edition-2023-spec>.

[Edition2024] Google, LLC, "Protocol Buffers Edition 2024 Language Specification", <https://protobuf.dev/reference/protobuf/edition-2024-spec>.

[MIMESNIFF] WHATWG, "MIME Sniffing - MIME Type Groups", WHATWG Living Standard, <https://mimesniff.spec.whatwg.org/#mime-type-groups>. Commit snapshot: <https://mimesniff.spec.whatwg.org/commit-snapshots/67bde18bc4f75a5c5b5dfb1217161137ce385124/>.

[Proto2] Google, LLC, "Protocol Buffers Language Specification (Proto2 Syntax)", <https://protobuf.dev/reference/protobuf/proto2-spec>.

[Proto3] Google, LLC, "Protocol Buffers Language Specification (Proto3)", <https://protobuf.dev/reference/protobuf/proto3-spec>.

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

[ProtoJSON] Google, LLC, "Protobuf JSON Wire Encoding Spec", <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/info/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/info/rfc8446>.

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

[UniChars] Bray, T. and P. Hoffman, "Unicode Character Repertoire Subsets", RFC 9839, DOI 10.17487/RFC9839, August 2025, <https://www.rfc-editor.org/info/rfc9839>.

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: varomodt@gmail.com