CoRE
Internet-Draft
Updates: 8428 (if approved)
Intended status: Standards Track
Expires: 6 December 2021

C. Bormann
Universitaet Bremen TZI

4 June 2021

SenML Features and Versions
draft-ietf-core-senml-versions-05

Abstract

This short document updates RFC 8428, Sensor Measurement Lists (SenML), by specifying the use of independently selectable "SenML Features" and mapping them to SenML version numbers.

Discussion Venues

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

Discussion of this document takes place on the CORE Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/ (https://mailarchive.ietf.org/arch/browse/core/).

Source for this draft and an issue tracker can be found at https://github.com/core-wg/senml-versions (https://github.com/core-wg/senml-versions).

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 6 December 2021.

Table of Contents

1. Introduction

The Sensor Measurement Lists (SenML) specification [RFC8428] provides a version number that is initially set to 10, without further specification on the way to make use of different version numbers.

The traditional idea of using a version number to indicate the evolution of an interchange format generally assumes an incremental progression of the version number as the format accretes additional features over time. However, in the case of SenML, it is expected that the likely evolution will be for independently selectable capability _features_ to be added to the basic specification that is indicated by version number 10. To support this model, this document repurposes the single version number accompanying a pack of SenML records so that it is interpreted as a bitmap that indicates the set of features a recipient would need to have implemented to be able to process the pack.

This short document specifies the use of SenML Features and maps them to SenML version number space, updating [RFC8428].

1.1. Terminology

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.

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C [C], including the "0b" prefix for binary numbers defined in Section 5.13.2 of the C++ language standard [Cplusplus], except that superscript notation (example for two to the power of 64: 2^64) denotes exponentiation; in the plain text version of this draft, superscript notation is rendered in paragraph text by C-incompatible surrogate notation as seen in this example, and in display math by a crude plaintext representation, as is the sum (Sigma) sign.

2. Feature Codes and the Version number

The present specification defines "SenML Features", each identified by a "feature name" (a text string) and a "feature code" (an unsigned integer less than 53).

The specific version of a SenML pack is composed of a set of features. The SenML version number ("bver" field) is then a bitmap of these features represented as an unsigned integer, specifically the sum of, for each feature present, two taken to the power of the feature code of that feature (Figure 1).

```
           __ 52                    fc
version = \         present(fc)  2
          /__ fc = 0
```

Figure 1: Feature bitmap as a sum of feature bits

where present(fc) is 1 if the feature with the feature code "fc" is present, 0 otherwise. (The expression 2^fc can be implemented as "1 << fc" in C and related languages.)

RFC editor: Please check that, in the TXT version, no " " crept into the above due to xml2rfc bug 641, and remove this paragraph. If possible with today's RFCXML, add the Sigma character as a parenthesis after "sum" in the caption.

2.1. Discussion

Representing features as a bitmap within a number is quite efficient as long as feature codes are sparingly allocated (see also Section 6).

Compatibility with the existing SenML version number, 10 decimal (0b1010), requires reserving four of the least significant bit positions for the base version as described in Section 3. There is an upper limit to the range of the integer numbers that can be represented in all SenML representations: practical JSON limits this to 2^53-1 [RFC7493]. This means the feature codes 4 to 52 are available, one of which is taken by the feature defined in Section 4, leaving 48 for allocation. (The current version 10 (with all other feature codes unset) can be visualized as "0b00000000000000000000000000000000000000000000001010".) For a lifetime of this scheme of several decades, approximately two feature codes per year or fewer should be allocated. Note that less generally applicable features can always be communicated via fields labeled with names that end with the "_" character ("must-understand fields"), see Section 4.4 of [RFC8428].)

Most representations visible to engineers working with SenML will use decimal numbers, e.g., 26 (0b11010, 0x1a) for a version that adds the "Secondary Units" feature (Section 4). This is slightly unwieldy, but will be quickly memorized in practice.

As a general observation, ending up over time with dozens of individually selectable optional extensions may lead to too many variants of what is supported by different implementations, reducing interoperability. So, in practice, it is still desirable to batch up extensions that are expected to be supported together into a single feature bit, leading to a sort of hybrid between completely independent extensions and a linear version scheme. This is also another reason why a space of 48 remaining feature codes should suffice for a while.

2.2. Updating Section 4.4 of [RFC8428]

The last paragraph of Section 4.4 of [RFC8428] may be read to give the impression that SenML version numbers are totally ordered, i.e., that an implementation that understands version n also always understands all versions k < n. If this ever was true for SenML versions before 10, it certainly is no longer true with this specification.

Any SenML pack that sets feature bits beyond the first four will lead to a version number that actually is greater than 10, so the requirement in Section 4.4 of [RFC8428] will prevent false interoperability with version 10 implementations.

Implementations that do implement feature bits beyond the first four, i.e., versions greater than 10, will instead need to perform a bitwise comparison of the feature bitmap as described in this specification and ensure that all features indicated are understood before using the pack. E.g., an implementation that implements basic SenML (version number 10) plus only a future feature code 5, will accept version number 42, but would not be able to work with a pack indicating version number 26 (base specification plus feature code 4). (If the implementation _requires_ feature code 5 without being backwards compatible, it will accept 42, but not 10.)

3. Features: Reserved0, Reserved1, Reserved2, Reserved3

For SenML Version 10 as described in [RFC8428], the feature codes 0 to 3 are already in use. Reserved1 (1) and Reserved3 (3) are always present and the features Reserved0 (0) and Reserved2 (2) are always absent, i.e., the four least significant bits set to 0b1010 indicate a version number of 10 if no other feature is in use. These four reserved feature codes are not to be used with any more specific semantics except in a specification that updates the present specification. (Note that Reserved0 and Reserved2 could be used in such a specification in a similar way to the way the feature codes 4 to 52 are in the present specification.)

4. Feature: Secondary Units

The feature "Secondary Units" (code number 4) indicates that secondary unit names [RFC8798] MAY be used in the "u" field of SenML Records, in addition to the primary unit names already allowed by [RFC8428].

Note that the most basic use of this feature simply sets the SenML version number to 26 (10 + 2^4).

5. Security Considerations

The security considerations of [RFC8428] apply. This specification provides structure to the interpretation of the SenML version number, which poses no additional security considerations except for some potential for surprise that version numbers do not simply increase linearly.

6. IANA Considerations

IANA is requested to create a new subregistry "SenML features" within the SenML registry [IANA.senml], with the registration policy "specification required" [RFC8126] and the columns:

* Feature code (an unsigned integer less than 53)
* Feature name (text)
* Specification

To facilitate the use of feature names in programs, the designated expert is requested to ensure that feature names are usable as identifiers in most programming languages, after lower-casing the feature name in the registry entry and replacing whitespace with underscores or hyphens, and that they also are distinct in this form.

The initial content of this registry is as follows:

Feature code	Feature name	Specification
0	Reserved0	RFCthis
1	Reserved1	RFCthis
2	Reserved2	RFCthis
3	Reserved3	RFCthis
4	Secondary Units	RFCthis, [RFC8798]

Table 1: Features defined for SenML at the time of writing

As the number of features that can be registered has a hard limit (48 codes left at the time of writing), the designated expert is specifically instructed to maintain a frugal regime of code point allocation, keeping code points available for SenML Features that are likely to be useful for non-trivial subsets of the SenML ecosystem. Quantitatively, the expert could for instance steer the allocation to a target of not allocating more than 10 % of the remaining set per year.

Where the specification of the feature code is provided in a document that is separate from the specification of the feature itself (as with feature code 4 above), both specifications should be listed.

7. References

7.1. Normative References

[C] International Organization for Standardization, "Information technology Programming languages C", ISO/IEC 9899:2018, Fourth Edition, June 2018, <https://www.iso.org/standard/74528.html>.

[Cplusplus] International Organization for Standardization, "Programming languages C++", ISO/IEC 14882:2020, Sixth Edition, December 2020, <https://www.iso.org/standard/79358.html>.

[IANA.senml] IANA, "Sensor Measurement Lists (SenML)", <http://www.iana.org/assignments/senml>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <https://www.rfc-editor.org/info/rfc8428>.

[RFC8798] Bormann, C., "Additional Units for Sensor Measurement Lists (SenML)", RFC 8798, DOI 10.17487/RFC8798, June 2020, <https://www.rfc-editor.org/info/rfc8798>.

7.2. Informative References

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <https://www.rfc-editor.org/info/rfc7493>.

Acknowledgements

Ari Keränen proposed to use the version number as a bitmap and provided further input on this specification. Jaime Jiménez helped clarify the document by providing a review. Elwyn Davies provided a detailed GENART review, with directly implementable text suggestions that now form part of this specification. Rob Wilton supplied comments one of which became the last paragraph of Section 2.1; Éric Vyncke helped with Section 2. Additional thanks go to the other IESG reviewers.

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org