Internet Engineering Task Force (IETF)
Request for Comments: 8428
Category: Standards Track
ISSN: 2070-1721

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson
C. Bormann
Universitaet Bremen TZI

August 2018

Sensor Measurement Lists (SenML)

Abstract

This specification defines a format for representing simple sensor measurements and device parameters in Sensor Measurement Lists (SenML). Representations are defined in JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR), Extensible Markup Language (XML), and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use one of these media types in protocols such as HTTP or the Constrained Application Protocol (CoAP) to transport the measurements of the sensor or to be configured.

Status of This Memo

Copyright Notice

Table of Contents

1. Overview

Connecting sensors to the Internet is not new, and there have been many protocols designed to facilitate it. This specification defines a format and media types for carrying simple sensor information in protocols such as HTTP [RFC7230] or CoAP [RFC7252]. The SenML format is designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time, a server parsing the data could collect a large number of sensor measurements in a relatively efficient manner. SenML can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and for the web resource model where the sensor data is requested as a resource representation (e.g., "GET /sensor/temperature").

There are many types of more complex measurements and measurements that this media type would not be suitable for. SenML strikes a balance between having some information about the sensor carried with the sensor data so that the data is self-describing, but it also tries to make that a fairly minimal set of auxiliary information for efficiency reasons. Other information about the sensor can be discovered by other methods such as using the Constrained RESTful Environments (CoRE) Link Format [RFC6690].

SenML is defined by a data model for measurements and simple metadata about measurements and devices. The data is structured as a single array that contains a series of SenML Records that can each contain fields such as a unique identifier for the sensor, the time the measurement was made, the unit the measurement is in, and the current value of the sensor. Serializations for this data model are defined for JSON [RFC8259], CBOR [RFC7049], XML [W3C.REC-xml-20081126], and Efficient XML Interchange (EXI) [W3C.REC-exi-20140211].

For example, the following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","v":23.1}
]
```

In the example above, the array has a single SenML Record with a measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a current value of 23.1 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets from large numbers of constrained devices. Keeping the total size of the payload small makes it easy to also use SenML in constrained networks, e.g., in an IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) [RFC4944]. It is always difficult to define what small code is, but there is a desire to be able to implement this in roughly 1 KB of flash on an 8-bit microprocessor. Experience with power meters and other large-scale deployments has indicated that the solution needs to support allowing multiple measurements to be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

The basic design is an array with a series of measurements. The following example shows two measurements made at different times. The value of a measurement is given by the "v" field, the time of a measurement is in the "t" field, the "n" field has a unique sensor name, and the unit of the measurement is carried in the "u" field.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,
   "v":23.5},
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020091e+09,
   "v":23.6}
]
```

To keep the messages small, it does not make sense to repeat the "n" field in each SenML Record, so there is a concept of a Base Name, which is simply a string that is prepended to the Name field of all elements in that Record and any Records that follow it. So, a more compact form of the example above is the following.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,
   "v":23.5},
  {"u":"Cel","t":1.276020091e+09,
   "v":23.6}
]
```

In the above example, the Base Name is in the "bn" field, and the "n" fields in each Record are empty strings, so they are omitted.

Some devices have accurate time while others do not, so SenML supports absolute and relative times. Time is represented in floating point as seconds. Values greater than or equal to 2**28 represent an absolute time relative to the Unix epoch. Values less than 2**28 represent time relative to the current time.

A simple sensor with no absolute wall-clock time might take a measurement every second, batch up 60 of them, and then send the batch to a server. It would include the relative time each measurement was made compared to the time the batch was sent in each SenML Record. If the server has accurate time based on, e.g., the Network Time Protocol (NTP), it may use the time it received the data and the relative offset to replace the times in the SenML with absolute times before saving the SenML information in a document database.

3. 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.

This document also uses the following terms:

SenML Record: One measurement or configuration instance in time presented using the SenML data model.

SenML Pack: One or more SenML Records in an array structure.

SenML Label: A short name used in SenML Records to denote different SenML fields (e.g., "v" for "value").

SenML Field: A component of a record that associates a value to a SenML Label for this record.

SenSML: Sensor Streaming Measurement List (see Section 4.8).

SenSML Stream: One or more SenML Records to be processed as a stream.

This document uses the terms "attribute" and "tag" where they occur with the underlying technologies (XML, CBOR [RFC7049], and the CoRE Link Format [RFC6690]); they are not used for SenML concepts, per se. However, note that "attribute" has been widely used in the past as a synonym for the SenML "field".

All comparisons of text strings are performed byte by byte, which results in the comparisons being case sensitive.

Where arithmetic is used, this specification uses the familiar notation of the programming language C, except that the operator "**" stands for exponentiation.

4. SenML Structure and Semantics

Each SenML Pack carries a single array that represents a set of measurements and/or parameters. This array contains a series of SenML Records with several fields described below. There are two kinds of fields: base and regular. Both the base and regular fields can be included in any SenML Record. The base fields apply to the entries in the Record and also to all Records after it up to, but not including, the next Record that has that same base field. All base fields are optional. Regular fields can be included in any SenML Record and apply only to that Record.

4.1. Base Fields

Base Name: This is a string that is prepended to the names found in the entries.

Base Time: A base time that is added to the time found in an entry.

Base Unit: A base unit that is assumed for all entries, unless otherwise indicated. If a record does not contain a Unit value, then the Base Unit is used. Otherwise, the value found in the Unit (if any) is used.

Base Value: A base value is added to the value found in an entry, similar to Base Time.

Base Sum: A base sum is added to the sum found in an entry, similar to Base Time.

Base Version: Version number of the media type format. This field is an optional positive integer and defaults to 10 if not present.

4.2. Regular Fields

Name: Name of the sensor or parameter. When appended to the Base Name field, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing, the Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

Unit: Unit for a measurement value. Optional.

Value: Value of the entry. Optional if a Sum value is present; otherwise, it’s required. Values are represented using basic data types. This specification defines floating-point numbers ("v" field for "Value"), booleans ("vb" for "Boolean Value"), strings ("vs" for "String Value"), and binary data ("vd" for "Data Value"). Exactly one Value field MUST appear unless there is a Sum field, in which case it is allowed to have no Value field.

Sum: Integrated sum of the values over time. Optional. This field is in the unit specified in the Unit value multiplied by seconds. For historical reasons, it is named "sum" instead of "integral".

Time: Time when the value was recorded. Optional.

Update Time: Period of time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. Optional. This can be used to detect the failure of sensors or the communications path from the sensor.

4.3. SenML Labels

Table 1 provides an overview of all SenML fields defined by this document with their respective labels and data types.

Name	Label	CBOR Label	JSON Type	XML Type
Base Name	bn	-2	String	string
Base Time	bt	-3	Number	double
Base Unit	bu	-4	String	string
Base Value	bv	-5	Number	double
Base Sum	bs	-6	Number	double
Base Version	bver	-1	Number	int
Name	n	0	String	string
Unit	u	1	String	string
Value	v	2	Number	double
String Value	vs	3	String	string
Boolean Value	vb	4	Boolean	boolean
Data Value	vd	8	String (*)	string (*)
Sum	s	5	Number	double
Time	t	6	Number	double
Update Time	ut	7	Number	double

Table 1: SenML Labels

(*) Data Value is a base64-encoded string with the URL-safe alphabet as defined in Section 5 of [RFC4648], with padding omitted. (In CBOR, the octets in the Data Value are encoded using a definite-length byte string, major type 2.)

For details of the JSON representation, see Section 5; for CBOR, see Section 6; and for XML, see Section 7.

4.4. Extensibility

The SenML format can be extended with further custom fields. Both new base and regular fields are allowed. See Section 12.2 for details. Implementations MUST ignore fields they don't recognize unless that field has a label name that ends with the "_" character, in which case an error MUST be generated.

All SenML Records in a Pack MUST have the same version number. This is typically done by adding a Base Version field to only the first Record in the Pack or by using the default value.

Systems reading one of the objects MUST check for the Base Version field. If this value is a version number larger than the version that the system understands, the system MUST NOT use this object. This allows the version number to indicate that the object contains structure or semantics that is different from what is defined in the present document beyond just making use of the extension points provided here. New version numbers can only be defined in an RFC that updates this specification or its successors.

4.5. Records and Their Fields

4.5.1. Names

The Name value is concatenated to the Base Name value to yield the name of the sensor. The resulting concatenated name needs to uniquely identify and differentiate the sensor from all others. The concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", and "0" to "9", as well as "-", ":", ".", "/", and "_"; furthermore, it MUST start with a character out of the set "A" to "Z", "a" to "z", or "0" to "9". This restricted character set was chosen so that concatenated names can be used directly within various URI schemes (including segments of an HTTP path with no special encoding; note that a name that contains "/" characters maps into multiple URI path segments) and can be used directly in many databases and analytic systems. [RFC5952] contains advice on encoding an IPv6 address in a name. See Section 14 for privacy considerations that apply to the use of long-term stable unique identifiers.

Although it is RECOMMENDED that concatenated names be represented as URIs [RFC3986] or URNs [RFC8141], the restricted character set specified above puts strict limits on the URI schemes and URN namespaces that can be used. As a result, implementers need to take care in choosing the naming scheme for concatenated names, because such names both need to be unique and need to conform to the restricted character set. One approach is to include a bit string that has guaranteed uniqueness (such as a 1-wire address [AN1796]). Some of the examples within this document use the device URN namespace as specified in [DEVICE-URN]. Universally Unique Identifiers (UUIDs) [RFC4122] are another way to generate a unique name. However, the restricted character set does not allow the use of many URI schemes, such as the "tag" scheme [RFC4151] and the "ni" scheme [RFC6920], in names as such. The use of URIs with characters incompatible with this set and possible mapping rules between the two are outside the scope of the present document.

4.5.2. Units

If the Record has no Unit, the Base Unit is used as the Unit. Having no Unit and no Base Unit is allowed; any information that may be required about units applicable to the value then needs to be provided by the application context.

4.5.3. Time

If either the Base Time or Time value is missing, the missing field is considered to have a value of zero. The Base Time and Time values are added together to get a value representing the time of measurement.

Values less than 268,435,456 (2**28) represent time relative to the current time. That is, a time of zero indicates that the sensor does not know the absolute time and the measurement was made roughly "now". A negative value indicates seconds in the past from roughly "now". Positive values up to 2**28 indicate seconds in the future from "now". An example for employing positive values would be actuation use, when the desired change should happen in the future, but the sender or the receiver does not have accurate time available.

Values greater than or equal to 2**28 represent an absolute time relative to the Unix epoch (1970-01-01T00:00Z in UTC time), and the time is counted the same way as the Portable Operating System Interface (POSIX) "seconds since the epoch" [TIME_T]. Therefore, the smallest absolute Time value that can be expressed (2**28) is 1978-07-04 21:24:16 UTC.

Because Time values up to 2**28 are used for representing time relative to "now" and Time and Base Time are added together, care must be taken to ensure that the sum does not inadvertently reach 2**28 (i.e., absolute time) when relative time was intended to be used.

Obviously, SenML Records referenced to "now" are only useful within a specific communication context (e.g., based on information on when the SenML Pack, or a specific Record in a SenSML Stream, was sent) or together with some other context information that can be used for deriving a meaning of "now"; the expectation for any archival use is that they will be processed into UTC-referenced records before that context would cease to be available. This specification deliberately leaves the accuracy of "now" very vague as it is determined by the overall systems that use SenML. In a system where a sensor without wall-clock time sends a SenML Record with a time referenced to "now" over a high-speed RS-485 link to an embedded system with accurate time that resolves "now" based on the time of reception, the resulting time uncertainty could be within 1 ms. At the other extreme, a deployment that sends SenML wind-speed readings over a Low-Earth Orbit (LEO) satellite link from a mountain valley might have resulting reception Time values that are easily a dozen minutes off the actual time of the sensor reading, with the time uncertainty depending on satellite locations and conditions.

4.5.4. Values

If only one of the Base Sum or Sum value is present, the missing field is considered to have a value of zero. The Base Sum and Sum values are added together to get the sum of measurement. If neither the Base Sum nor the Sum is present, then the measurement does not have a Sum value.

If the Base Value or Value is not present, the missing field(s) is considered to have a value of zero. The Base Value and Value are added together to get the value of the measurement.

Representing the statistical characteristics of measurements, such as accuracy, can be very complex. Future specification may add new fields to provide better information about the statistical properties of the measurement.

In summary, the structure of a SenML Record is laid out to support a single measurement per Record. If multiple data values are measured at the same time (e.g., air pressure and altitude), they are best kept as separate Records linked through their Time value; this is even true when one of the data values is more "meta" than others (e.g., describes a condition that influences other measurements at the same time).

4.6. Resolved Records

Sometimes it is useful to be able to refer to a defined normalized format for SenML Records. This normalized format tends to get used for big data applications and intermediate forms when converting to other formats. Also, if SenML Records are used outside of a SenML Pack, they need to be resolved first to ensure applicable base values are applied.

A SenML Record is referred to as "resolved" if it does not contain any base values, i.e., labels starting with the character "b", except for Base Version fields (see below), and has no relative times. To resolve the Records, the applicable base values of the SenML Pack (if any) are applied to the Record. That is, for the base values in the Record or before the Record in the Pack, Name and Base Name are concatenated, the Base Time is added to the time of the Record, the Base Unit is applied to the Record if it did not contain a Unit, etc. In addition, the Records need to be in chronological order in the Pack. An example of this is shown in Section 5.1.4.

The Base Version field MUST NOT be present in resolved Records if the SenML version defined in this document is used; otherwise, it MUST be present in all the resolved SenML Records.

A future specification that defines new base fields needs to specify how the field is resolved.

4.7. Associating Metadata

SenML is designed to carry the minimum dynamic information about measurements and, for efficiency reasons, does not carry significant static metadata about the device, object, or sensors. Instead, it is assumed that this metadata is carried out of band. For web resources using SenML Packs, this metadata can be made available using the CoRE Link Format [RFC6690]. The most obvious use of this link format is to describe that a resource is available in a SenML format in the first place. The relevant media type indicator is included in the Content-Type (ct=) link attribute (which is defined for the link format in Section 7.2.1 of [RFC7252]).

4.8. Sensor Streaming Measurement Lists (SenSML)

In some usage scenarios of SenML, the implementations store or transmit SenML in a stream-like fashion, where data is collected over time and continuously added to the object. This mode of operation is optional, but systems or protocols using SenML in this fashion MUST specify that they are doing this. SenML defines separate media types to indicate Sensor Streaming Measurement Lists (SenSML) for this usage (see Section 12.3.2). In this situation, the SenSML Stream can be sent and received in a partial fashion, i.e., a measurement entry can be read as soon as the SenML Record is received and does not have to wait for the full SenSML Stream to be complete.

If times relative to "now" (see Section 4.5.3) are used in SenML Records of a SenSML Stream, their interpretation of "now" is based on the time when the specific Record is sent in the stream.

4.9. Configuration and Actuation Usage

SenML can also be used for configuring parameters and controlling actuators. When a SenML Pack is sent (e.g., using an HTTP/CoAP POST or PUT method) and the semantics of the target are such that SenML is interpreted as configuration/actuation, SenML Records are interpreted as a request to change the values of given (sub)resources (given as names) to given values at the given time(s). The semantics of the target resource supporting this usage can be described, e.g., using [RID-CoRE]. Examples of actuation usage are shown in Section 5.1.7.

5. JSON Representation (application/senml+json)

For the SenML fields shown in Table 2, the SenML Labels are used as the JSON object member names within JSON objects representing the JSON SenML Records.

Name	Label	JSON Type
Base Name	bn	String
Base Time	bt	Number
Base Unit	bu	String
Base Value	bv	Number
Base Sum	bs	Number
Base Version	bver	Number
Name	n	String
Unit	u	String
Value	v	Number
String Value	vs	String
Boolean Value	vb	Boolean
Data Value	vd	String
Sum	s	Number
Time	t	Number
Update Time	ut	Number

Table 2: JSON SenML Labels

The root JSON value consists of an array with one JSON object for each SenML Record. All the fields in the above table MAY occur in the Records with member values of the type specified in the table.

Only the UTF-8 [RFC3629] form of JSON is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC8259]. Octets in the Data Value are base64 encoded with the URL-safe alphabet as defined in Section 5 of [RFC4648], with padding omitted.

Systems receiving measurements MUST be able to process the range of floating-point numbers that are representable as IEEE double-precision, floating-point numbers [IEEE.754]. This allows Time values to have better than microsecond precision over the next 100 years. The number of significant digits in any measurement is not relevant, so a reading of 1.1 has exactly the same semantic meaning as 1.10. If the value has an exponent, the "e" MUST be in lower case. In the interest of avoiding unnecessary verbosity and speeding up processing, the mantissa SHOULD be less than 19 characters long, and the exponent SHOULD be less than 5 characters long.

5.1. Examples

5.1.1. Single Data Point

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","v":23.1}
]
```

5.1.2. Multiple Data Points

The following example shows voltage and current "now", i.e., at an unspecified time.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:","n":"voltage","u":"V","v":120.1},
  {"n":"current","u":"A","v":1.2}
]
```

The next example is similar to the above one, but it shows current at Tue Jun 8 18:01:16.001 UTC 2010 and at each second for the previous 5 seconds.

```
[
  {"bn":"urn:dev:ow:10e2073a0108006:","bt":1.276020076001e+09,
   "bu":"A","bver":5,
   "n":"voltage","u":"V","v":120.1},
  {"n":"current","t":-5,"v":1.2},
  {"n":"current","t":-4,"v":1.3},
  {"n":"current","t":-3,"v":1.4},
  {"n":"current","t":-2,"v":1.5},
  {"n":"current","t":-1,"v":1.6},
  {"n":"current","v":1.7}
]
```

As an example of SenSML, the following stream of measurements may be sent via a long-lived HTTP POST from the producer of the stream to its consumer, and each measurement object may be reported at the time it was measured:

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","bt":1.320067464e+09,
   "bu":"%RH","v":21.2},
  {"t":10,"v":21.3},
  {"t":20,"v":21.4},
  {"t":30,"v":21.4},
  {"t":40,"v":21.5},
  {"t":50,"v":21.5},
  {"t":60,"v":21.5},
  {"t":70,"v":21.6},
  {"t":80,"v":21.7},
...
```

5.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with a 1-wire address 10e2073a01080063, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provides position data, which is provided in the same measurement or parameter array as separate entries. Note that time is used to correlate data that belongs together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063","bt":1.320067464e+09,
   "bu":"%RH","v":20},
  {"u":"lon","v":24.30621},
  {"u":"lat","v":60.07965},
  {"t":60,"v":20.3},
  {"u":"lon","t":60,"v":24.30622},
  {"u":"lat","t":60,"v":60.07965},
  {"t":120,"v":20.7},
  {"u":"lon","t":120,"v":24.30623},
  {"u":"lat","t":120,"v":60.07966},
  {"u":"%EL","t":150,"v":98},
  {"t":180,"v":21.2},
  {"u":"lon","t":180,"v":24.30628},
  {"u":"lat","t":180,"v":60.07967}
]
```

The following table shows the size of this example in various forms, as well as the size of each of these forms compressed with gzip.

Encoding	Size	Compressed Size
JSON	573	206
XML	649	235
CBOR	254	196
EXI	161	184

Table 3: Size Comparisons

5.1.4. Resolved Data

The following shows the example from the previous section in resolved format.

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067464e+09,
   "v":20},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067464e+09,
   "v":24.30621},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067464e+09,
   "v":60.07965},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067524e+09,
   "v":20.3},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067524e+09,
   "v":24.30622},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067524e+09,
   "v":60.07965},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067584e+09,
   "v":20.7},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067584e+09,
   "v":24.30623},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067584e+09,
   "v":60.07966},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%EL","t":1.320067614e+09,
   "v":98},
  {"n":"urn:dev:ow:10e2073a01080063","u":"%RH","t":1.320067644e+09,
   "v":21.2},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lon","t":1.320067644e+09,
   "v":24.30628},
  {"n":"urn:dev:ow:10e2073a01080063","u":"lat","t":1.320067644e+09,
   "v":60.07967}
]
```

5.1.5. Multiple Data Types

The following example shows a sensor that returns different data types.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:","n":"temp","u":"Cel","v":23.1},
  {"n":"label","vs":"Machine Room"},
  {"n":"open","vb":false},
  {"n":"nfc-reader","vd":"aGkgCg"}
]
```

5.1.6. Collection of Resources

The following example shows the results from a query to one device that aggregates multiple measurements from other devices. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on http://[2001:db8::2] at Mon Oct 31 16:27:09 UTC 2011 and has gotten two separate values as a result: a temperature and humidity measurement as well as the results from another device at http://[2001:db8::1] that also had a temperature and humidity measurement. Note that the last record would use the Base Name from the 3rd record but the Base Time from the first record.

```
[
  {"bn":"2001:db8::2/","bt":1.320078429e+09,
   "n":"temperature","u":"Cel","v":25.2},
  {"n":"humidity","u":"%RH","v":30},
  {"bn":"2001:db8::1/","n":"temperature","u":"Cel","v":12.3},
  {"n":"humidity","u":"%RH","v":67}
]
```

5.1.7. Setting an Actuator

The following example shows the SenML that could be used to set the current set point of a typical residential thermostat that has a temperature set point, a switch to turn on and off the heat, and a switch to turn on the fan override.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:"},
  {"n":"temp","u":"Cel","v":23.1},
  {"n":"heat","u":"/","v":1},
  {"n":"fan","u":"/","v":0}
]
```

In the following example, two different lights are turned on. It is assumed that the lights are on a network that can guarantee delivery of the messages to the two lights within 15 ms (e.g., a network using 802.1BA [IEEE802.1BA] and 802.1AS [IEEE802.1AS] for time synchronization). The controller has set the time of the lights to come on at 20 ms in the future from the current time. This allows both lights to receive the message, wait till that time, then apply the switch command so that both lights come on at the same time.

```
[
  {"bt":1.320078429e+09,"bu":"/","n":"2001:db8::3","v":1},
  {"n":"2001:db8::4","v":1}
]
```

The following shows two lights being turned off using a non-deterministic network that has high odds of delivering a message in less than 100 ms and uses NTP for time synchronization. The current time is 1320078429. The user has just turned off a light switch that is turning off two lights. Both lights are immediately dimmed to 50% brightness to give the user instant feedback that something is changing. However, given the network, the lights will probably dim at somewhat different times. Then 100 ms in the future, both lights will go off at the same time. The instant, but not synchronized, dimming gives the user the sensation of quick responses, and the timed-off 100 ms in the future gives the perception of both lights going off at the same time.

```
[
  {"bt":1.320078429e+09,"bu":"/","n":"2001:db8::3","v":0.5},
  {"n":"2001:db8::4","v":0.5},
  {"n":"2001:db8::3","t":0.1,"v":0},
  {"n":"2001:db8::4","t":0.1,"v":0}
]
```

6. CBOR Representation (application/senml+cbor)

The CBOR [RFC7049] representation is equivalent to the JSON representation, with the following changes:

- For JSON Numbers, the CBOR representation can use integers, floating-point numbers, or decimal fractions (CBOR Tag 4); however, a representation SHOULD be chosen such that when the CBOR value is converted to an IEEE double-precision, floating-point value, it has exactly the same value as the original JSON Number converted to that form. For the version number, only an unsigned integer is allowed.

- Characters in the String Value are encoded using a text string with a definite length (major type 3). Octets in the Data Value are encoded using a byte string with a definite length (major type 2).

- For compactness, the CBOR representation uses integers for the labels, as defined in Table 4. This table is conclusive, i.e., there is no intention to define any additional integer map keys; any extensions will use string map keys. This allows translators converting between CBOR and JSON representations to also convert all future labels without needing to update implementations. Base values are given negative CBOR labels, and others are given non-negative labels.

Name	Label	CBOR Label
Base Version	bver	-1
Base Name	bn	-2
Base Time	bt	-3
Base Unit	bu	-4
Base Value	bv	-5
Base Sum	bs	-6
Name	n	0
Unit	u	1
Value	v	2
String Value	vs	3
Boolean Value	vb	4
Sum	s	5
Time	t	6
Update Time	ut	7
Data Value	vd	8

Table 4: CBOR Representation: Integers for Map Keys

- o For streaming SenSML in CBOR representation, the array containing the records SHOULD be a CBOR array with an indefinite length; for non-streaming SenML, an array with a definite length MUST be used.

The following example shows a dump of the CBOR example for the same sensor measurement as in Section 5.1.2.

```
0000 87 a7 21 78 1b 75 72 6e 3a 64 65 76 3a 6f 77 3a |..!x.urn:dev:ow:|
0010 31 30 65 32 30 37 33 61 30 31 30 38 30 30 36 3a |10e2073a0108006:|
0020 22 fb 41 d3 03 a1 5b 00 10 62 23 61 41 20 05 00 |".A...[..b#aA ..|
0030 67 76 6f 6c 74 61 67 65 01 61 56 02 fb 40 5e 06 |gvoltage.aV..@^.|
0040 66 66 66 66 66 a3 00 67 63 75 72 72 65 6e 74 06 |fffff..gcurrent.|
0050 24 02 fb 3f f3 33 33 33 33 33 33 a3 00 67 63 75 |$..?.333333..gcu|
0060 72 72 65 6e 74 06 23 02 fb 3f f4 cc cc cc cc cc |rrent.#..?......|
0070 cd a3 00 67 63 75 72 72 65 6e 74 06 22 02 fb 3f |...gcurrent."..?|
0080 f6 66 66 66 66 66 66 a3 00 67 63 75 72 72 65 6e |.ffffff..gcurren|
0090 74 06 21 02 f9 3e 00 a3 00 67 63 75 72 72 65 6e |t.!..>...gcurren|
00a0 74 06 20 02 fb 3f f9 99 99 99 99 99 9a a3 00 67 |t. ..?.........g|
00b0 63 75 72 72 65 6e 74 06 00 02 fb 3f fb 33 33 33 |current....?.333|
00c0 33 33 33                                        |333|
00c3
```

In CBOR diagnostic notation (Section 6 of [RFC7049]), this is:

```
[{-2: "urn:dev:ow:10e2073a0108006:",
  -3: 1276020076.001, -4: "A", -1: 5, 0: "voltage", 1: "V", 2: 120.1},
 {0: "current", 6: -5, 2: 1.2}, {0: "current", 6: -4, 2: 1.3},
 {0: "current", 6: -3, 2: 1.4}, {0: "current", 6: -2, 2: 1.5},
 {0: "current", 6: -1, 2: 1.6}, {0: "current", 6: 0, 2: 1.7}]
```

7. XML Representation (application/senml+xml)

A SenML Pack or Stream can also be represented in XML format as defined in this section.

Only the UTF-8 form of XML is allowed. Octets in the Data Value are base64 encoded with the URL-safe alphabet as defined in Section 5 of [RFC4648], with padding omitted.

The following shows an XML example for the same sensor measurement as in Section 5.1.2.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a0108006:" bt="1.276020076001e+09"
  bu="A" bver="5" n="voltage" u="V" v="120.1"></senml>
  <senml n="current" t="-5" v="1.2"></senml>
  <senml n="current" t="-4" v="1.3"></senml>
  <senml n="current" t="-3" v="1.4"></senml>
  <senml n="current" t="-2" v="1.5"></senml>
  <senml n="current" t="-1" v="1.6"></senml>
  <senml n="current" v="1.7"></senml>
</sensml>
```

The SenML Stream is represented as a sensml element that contains a series of senml elements for each SenML Record. The SenML fields are represented as XML attributes. For each field defined in this document, the following table shows the SenML Labels, which are used for the XML attribute name, as well as the according restrictions on the XML attribute values ("type") as used in the XML senml elements.

Name	Label	XML Type
Base Name	bn	string
Base Time	bt	double
Base Unit	bu	string
Base Value	bv	double
Base Sum	bs	double
Base Version	bver	int
Name	n	string
Unit	u	string
Value	v	double
String Value	vs	string
Data Value	vd	string
Boolean Value	vb	boolean
Sum	s	double
Time	t	double
Update Time	ut	double

Table 5: XML SenML Labels

The RelaxNG [RNC] Schema for the XML is:

```
default namespace = "urn:ietf:params:xml:ns:senml"
namespace rng = "http://relaxng.org/ns/structure/1.0"

senml = element senml {
  attribute bn { xsd:string }?,
  attribute bt { xsd:double }?,
  attribute bv { xsd:double }?,
  attribute bs { xsd:double }?,
  attribute bu { xsd:string }?,
  attribute bver { xsd:int }?,

  attribute n { xsd:string }?,
  attribute s { xsd:double }?,
  attribute t { xsd:double }?,
  attribute u { xsd:string }?,
  attribute ut { xsd:double }?,

  attribute v { xsd:double }?,
  attribute vb { xsd:boolean }?,
  attribute vs { xsd:string }?,
  attribute vd { xsd:string }?
}
```

```
   sensml =
     element sensml {
       senml+
   }

   start = sensml
```

8. EXI Representation (application/senml-exi)

For efficient transmission of SenML over, e.g., a constrained network, EXI can be used. This encodes the XML Schema [W3C.REC-xmlschema-1-20041028] structure of SenML into binary tags and values rather than ASCII text. An EXI representation of SenML SHOULD be made using the strict schema mode of EXI. However, this mode does not allow tag extensions to the schema; therefore, any extensions will be lost in the encoding. For uses where extensions need to be preserved in EXI, the non-strict schema mode of EXI MAY be used.

The EXI header MUST include "EXI Options", as defined in [W3C.REC-exi-20140211], with a schemaId set to the value of "a", indicating the schema provided in this specification. Future revisions to the schema can change the value of the schemaId to allow for backwards compatibility. When the data will be transported over CoAP or HTTP, an EXI Cookie SHOULD NOT be used as it simply makes things larger and is redundant to information provided in the Content-Type header.

The following is the XSD Schema to be used for strict schema-guided EXI processing. It is generated from the RelaxNG.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="urn:ietf:params:xml:ns:senml"
xmlns:ns1="urn:ietf:params:xml:ns:senml">
  <xs:element name="senml">
    <xs:complexType>
      <xs:attribute name="bn" type="xs:string" />
      <xs:attribute name="bt" type="xs:double" />
      <xs:attribute name="bv" type="xs:double" />
      <xs:attribute name="bs" type="xs:double" />
      <xs:attribute name="bu" type="xs:string" />
      <xs:attribute name="bver" type="xs:int" />
      <xs:attribute name="n" type="xs:string" />
      <xs:attribute name="s" type="xs:double" />
      <xs:attribute name="t" type="xs:double" />
      <xs:attribute name="u" type="xs:string" />
      <xs:attribute name="ut" type="xs:double" />
      <xs:attribute name="v" type="xs:double" />
      <xs:attribute name="vb" type="xs:boolean" />
      <xs:attribute name="vs" type="xs:string" />
      <xs:attribute name="vd" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="sensml">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="ns1:senml" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note that this example is the same information as the first example in Section 5.1.2 but in JSON format.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a01080063:" n="voltage" u="V"
  v="120.1"></senml>
  <senml n="current" u="A" v="1.2"></senml>
</sensml>
```

Which compresses with EXI to the following displayed in hexdump:

```
0000 a0 30 0d 84 80 f3 ab 93 71 d3 23 2b b1 d3 7b b9 |.0......q.#+..{.|
0010 d1 89 83 29 91 81 b9 9b 09 81 89 81 c1 81 81 b1 |...)............|
0020 99 d2 84 bb 37 b6 3a 30 b3 b2 90 1a b1 58 84 c0 |....7.:0.....X..|
0030 33 04 b1 ba b9 39 32 b7 3a 10 1a 09 06 40 38    |3....92.:....@8|
003f
```

The above example used the bit-packed form of EXI, but it is also possible to use a byte-packed form of EXI, which can make it easier for a simple sensor to produce valid EXI without really implementing EXI. Consider the example of a temperature sensor that produces a value in tenths of degrees Celsius over a range of 0.0 to 55.0. It would produce an XML SenML file such as:

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml n="urn:dev:ow:10e2073a01080063" u="Cel" v="23.1"></senml>
</sensml>
```

The compressed form, using the byte-alignment option of EXI, for the above XML is the following:

```
0000 a0 00 48 80 6c 20 01 06 1d 75 72 6e 3a 64 65 76 |..H.l ...urn:dev|
0010 3a 6f 77 3a 31 30 65 32 30 37 33 61 30 31 30 38 |:ow:10e2073a0108|
0020 30 30 36 33 02 05 43 65 6c 01 00 e7 01 01 00 03 |0063..Cel.......|
0030 01                                              |.|
0031
```

A small temperature sensor device that only generates this one EXI file does not really need a full EXI implementation. It can simply hard code the output, replacing the 1-wire device ID starting at byte 0x14 and going to byte 0x23 with its device ID and replacing the value "0xe7 0x01" at location 0x2b and 0x2c with the current temperature. The EXI specification [W3C.REC-exi-20140211] contains the full information on how floating-point numbers are represented, but for the purpose of this sensor, the temperature can be converted to an integer in tenths of degrees (231 in this example). EXI stores 7 bits of the integer in each byte with the top bit set to one if there are further bytes. So, the first byte is set to the low 7 bits of the integer temperature in tenths of degrees plus 0x80. In this example, 231 & 0x7F + 0x80 = 0xE7. The second byte is set to the integer temperature in tenths of degrees right-shifted 7 bits. In this example, 231 >> 7 = 0x01.

9. Fragment Identification Methods

A SenML Pack typically consists of multiple SenML Records, and for some applications, it may be useful to be able to refer to a single Record, or a set of Records, in a Pack with a fragment identifier. The fragment identifier is only interpreted by a client and does not impact retrieval of a representation. The SenML fragment identification is modeled after Comma-Separated Value (CSV) fragment identifiers [RFC7111].

To select a single SenML Record, the "rec" scheme followed by a single number is used. For the purpose of numbering Records, the first Record is at position 1. A range of records can be selected by giving the first and the last record number separated by a "-" character. Instead of the second number, the "*" character can be used to indicate the last SenML Record in the Pack. A set of Records can also be selected using a comma-separated list of Record positions or ranges.

(We use the term "selecting a Record" for identifying it as part of the fragment, not in the sense of isolating it from the Pack -- the Record still needs to be interpreted as part of the Pack, e.g., using the base values defined in earlier Records.)

9.1. Fragment Identification Examples

The 3rd SenML Record from the "coap://example.com/temp" resource can be selected with:

```
coap://example.com/temp#rec=3
```

Records from 3rd to 6th can be selected with:

```
coap://example.com/temp#rec=3-6
```

Records from 19th to the last can be selected with:

```
coap://example.com/temp#rec=19-*
```

The 3rd and 5th Records can be selected with:

```
coap://example.com/temp#rec=3,5
```

To select the Records from third to fifth, the 10th Record, and all Records from 19th to the last:

```
coap://example.com/temp#rec=3-5,10,19-*
```

9.2. Fragment Identification for XML and EXI Formats

In addition to the SenML fragment identifiers described above, with the XML and EXI SenML formats, the syntax defined in the XPointer element() Scheme [XPointerElement] of the XPointer Framework [XPointerFramework] can be used. (This is required by [RFC7303] for media types using the syntax suffix structured with "+xml". For consistency, SenML allows this for the EXI formats as well.)

Note that fragment identifiers are available to the client side only; they are not provided in transfer protocols such as CoAP or HTTP. Thus, they cannot be used by the server in deciding which media type to send. Where a server has multiple representations available for a resource identified by a URI, it might send a JSON or CBOR representation when the client was directed to use an XML/EXI fragment identifier with it. Clients can prevent running into this problem by explicitly requesting an XML or EXI media type (e.g., using the CoAP Accept option) when XML-/EXI-only fragment identifier syntax is in use in the URI.

10. Usage Considerations

The measurements support sending both the current value of a sensor as well as an integrated sum. For many types of measurements, the sum is more useful than the current value. For historical reasons, this field is called "Sum" instead of "integral", which would more accurately describe its function. For example, an electrical meter that measures the energy a given computer uses will typically want to measure the cumulative amount of energy used. This is less prone to error than reporting the power each second and trying to have something on the server side sum together all the power measurements. If the network between the sensor and the meter goes down over some period of time, when it comes back up, the cumulative sum helps reflect what happened while the network was down. A meter like this would typically report a measurement with the unit set to watts, but it would put the sum of energy used in the "s" field of the measurement. It might optionally include the current power in the "v" field.

While the benefit of using the integrated sum is fairly clear for measurements like power and energy, it is less obvious for something like temperature. Reporting the sum of the temperature makes it easy to compute averages even when the individual temperature values are not reported frequently enough to compute accurate averages. Implementers are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative Sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation, they may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.

2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.

3. Applications cannot make assumptions about when the device started accumulating values into the sum.

Typically, applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are non-negative, the sum should never get smaller; if the sum does get smaller, the application will know that one of the situations listed above has happened.

Despite the name "Sum", the Sum field is not useful for applications that maintain a running count of the number of times an event happened or that keep track of a counter such as the total number of bytes sent on an interface. Data like that can be sent directly in the Value field.

11. CDDL

As a convenient reference, the JSON and CBOR representations can be described with the common Concise Data Definition Language (CDDL) specification [CDDL-CBOR] in Figure 1 (informative).

```
SenML-Pack = [1* record]

record = {
  ? bn => tstr,        ; Base Name
  ? bt => numeric,     ; Base Time
  ? bu => tstr,        ; Base Units
  ? bv => numeric,     ; Base Value
  ? bs => numeric,     ; Base Sum
  ? bver => uint,      ; Base Version
  ? n => tstr,        ; Name
  ? u => tstr,        ; Units
  ? s => numeric,     ; Sum
  ? t => numeric,     ; Time
  ? ut => numeric,    ; Update Time
  ? ( v => numeric // ; Numeric Value
      vs => tstr //   ; String Value
      vb => bool //   ; Boolean Value
      vd => binary-value ) ; Data Value
  * key-value-pair
}

; now define the generic versions
key-value-pair = ( label => value )

label = non-b-label / b-label
non-b-label = tstr .regexp  "[A-Zac-z0-9][-_:.A-Za-z0-9]*" / uint
b-label = tstr .regexp  "b[-_:.A-Za-z0-9]+" / nint

value = tstr / binary-value / numeric / bool
numeric = number / decfrac
```

Figure 1: Common CDDL Specification for CBOR and JSON SenML

For JSON, we use text labels and base64url-encoded binary data (Figure 2).

```
bver = "bver" n  = "n"   s  = "s"
bn   = "bn"   u  = "u"   t  = "t"
bt   = "bt"   v  = "v"   ut = "ut"
bu   = "bu"   vs = "vs"  vd = "vd"
bv   = "bv"   vb = "vb"
bs   = "bs"

binary-value = tstr             ; base64url encoded
```

Figure 2: JSON-Specific CDDL Specification for SenML

For CBOR, we use integer labels and native binary data (Figure 3).

```
bver = -1  n  = 0   s  = 5
bn   = -2  u  = 1   t  = 6
bt   = -3  v  = 2   ut = 7
bu   = -4  vs = 3   vd = 8
bv   = -5  vb = 4
bs   = -6

binary-value = bstr
```

Figure 3: CBOR-Specific CDDL Specification for SenML

12. IANA Considerations

IANA has created a new "Sensor Measurement Lists (SenML)" registry that contains the subregistries defined in Sections 12.1 and 12.2.

12.1. SenML Units Registry

IANA has created a registry of SenML unit symbols called the "SenML Units" registry. The primary purpose of this registry is to make sure that symbols uniquely map to indicate a type of measurement. Definitions for many of these units can be found in other publications such as [NIST811] and [BIPM]. Units marked with an asterisk are NOT RECOMMENDED to be produced by new implementations but are in active use and SHOULD be implemented by consumers that can use the corresponding SenML units that are closer to the unscaled SI units.

Symbol	Description	Type	Reference
m	meter	float	RFC 8428
kg	kilogram	float	RFC 8428
g	gram*	float	RFC 8428
s	second	float	RFC 8428
A	ampere	float	RFC 8428
K	kelvin	float	RFC 8428
cd	candela	float	RFC 8428
mol	mole	float	RFC 8428
Hz	hertz	float	RFC 8428
rad	radian	float	RFC 8428
sr	steradian	float	RFC 8428
N	newton	float	RFC 8428
Pa	pascal	float	RFC 8428
J	joule	float	RFC 8428
W	watt	float	RFC 8428
C	coulomb	float	RFC 8428
V	volt	float	RFC 8428
F	farad	float	RFC 8428
Ohm	ohm	float	RFC 8428
S	siemens	float	RFC 8428
Wb	weber	float	RFC 8428
T	tesla	float	RFC 8428
H	henry	float	RFC 8428
Cel	degrees Celsius	float	RFC 8428
lm	lumen	float	RFC 8428
lx	lux	float	RFC 8428
Bq	becquerel	float	RFC 8428
Gy	gray	float	RFC 8428
Sv	sievert	float	RFC 8428
kat	katal	float	RFC 8428
m2	square meter (area)	float	RFC 8428
m3	cubic meter (volume)	float	RFC 8428
l	liter (volume)*	float	RFC 8428
m/s	meter per second (velocity)	float	RFC 8428
m/s2	meter per square second (acceleration)	float	RFC 8428
m3/s	cubic meter per second (flow rate)	float	RFC 8428
l/s	liter per second (flow rate)*	float	RFC 8428
W/m2	watt per square meter (irradiance)	float	RFC 8428
cd/m2	candela per square meter (luminance)	float	RFC 8428
bit	bit (information content)	float	RFC 8428
bit/s	bit per second (data rate)	float	RFC 8428
lat	degrees latitude (Note 1)	float	RFC 8428
lon	degrees longitude (Note 1)	float	RFC 8428

pH	pH value (acidity; logarithmic quantity)	float	RFC 8428
dB	decibel (logarithmic quantity)	float	RFC 8428
dBW	decibel relative to 1 W (power level)	float	RFC 8428
Bspl	bel (sound pressure level; logarithmic quantity)*	float	RFC 8428
count	1 (counter value)	float	RFC 8428
/	1 (ratio, e.g., value of a switch; Note 2)	float	RFC 8428
%	1 (ratio, e.g., value of a switch; Note 2)*	float	RFC 8428
%RH	percentage (relative humidity)	float	RFC 8428
%EL	percentage (remaining battery energy level)	float	RFC 8428
EL	seconds (remaining battery energy level)	float	RFC 8428
1/s	1 per second (event rate)	float	RFC 8428
1/min	1 per minute (event rate, "rpm")*	float	RFC 8428
beat/min	1 per minute (heart rate in beats per minute)*	float	RFC 8428
beats	1 (cumulative number of heart beats)*	float	RFC 8428
S/m	siemens per meter (conductivity)	float	RFC 8428

Table 6: IANA Registry for SenML Units

o Note 1: Assumed to be in World Geodetic System 1984 (WGS84), unless another reference frame is known for the sensor.

o Note 2: A value of 0.0 indicates the switch is off, 1.0 indicates on, and 0.5 indicates half on. The preferred name of this unit is "/". For historical reasons, the name "%" is also provided for the same unit, but note that while that name strongly suggests a percentage (0..100), it is NOT a percentage but the absolute ratio!

New entries can be added to the registration by Expert Review as defined in [RFC8126]. Experts should exercise their own good judgment but need to consider the following guidelines:

1. There needs to be a real and compelling use for any new unit to be added.

2. Each unit should define the semantic information and be chosen carefully. Implementers need to remember that the same word may be used in different real-life contexts. For example, degrees when measuring latitude have no semantic relation to degrees when measuring temperature; thus, two different units are needed.

3. These measurements are produced by computers for consumption by computers. The principle is that conversion has to be easily done when both reading and writing the media type. The value of a single canonical representation outweighs the convenience of easy human representations or loss of precision in a conversion.

4. Use of System of Units (SI) prefixes such as "k" before the unit is not recommended. Instead, one can represent the value using scientific notation such as 1.2e3. The "kg" unit is an exception to this rule since it is an SI base unit; the "g" unit is provided for legacy compatibility.

5. For a given type of measurement, there will only be one unit type defined. So for length, meter is defined, and other lengths such as mile, foot, and light year are not allowed. For most cases, the SI unit is preferred.

   (Note that some amount of judgment will be required here, as even SI itself is not entirely consistent in this respect. For instance, for temperature, [ISO-80000-5] defines a quantity, item 5-1 (thermodynamic temperature), and a corresponding unit of 5-1.a (Kelvin); [ISO-80000-5] goes on to define another quantity, item 5-2 ("Celsius temperature"), and the corresponding unit of 5-2.a (degree Celsius). The latter quantity is defined such that it gives the thermodynamic temperature as a delta from T0 = 275.15 K. ISO 80000-5 is defining both units side by side and not really expressing a preference. This level of recognition of the alternative unit degree Celsius is the reason why Celsius temperatures seem exceptionally acceptable in the SenML units list alongside Kelvin.)

6. Symbol names that could be easily confused with existing common units or units combined with prefixes should be avoided. For example, selecting a unit name of "mph" to indicate something that had nothing to do with velocity would be a bad choice, as "mph" is commonly used to mean "miles per hour".

7. The following should not be used because they are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, u, n, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, and Yi.

8. The following units should not be used as they are commonly used to represent other measurements: Ky, Gal, dyn, etg, P, St, Mx, G, Oe, Gb, sb, Lmb, mph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, and kWh.

9. The unit names are case sensitive, and the correct case needs to be used; however, symbols that differ only in case should not be allocated.

10. A number after a unit typically indicates the previous unit raised to that power, and "/" indicates that the units that follow are the reciprocals. A unit should have only one "/" in the name.

11. A good list of common units can be found in the Unified Code for Units of Measure [UCUM].

12.2. SenML Labels Registry

IANA has created a new registry for SenML Labels called the "SenML Labels" registry. The initial contents of the registry are as follows:

Name	Label	CL	JSON Type	XML Type	EI	Reference
Base Name	bn	-2	String	string	a	RFC 8428
Base Time	bt	-3	Number	double	a	RFC 8428
Base Unit	bu	-4	String	string	a	RFC 8428
Base Value	bv	-5	Number	double	a	RFC 8428
Base Sum	bs	-6	Number	double	a	RFC 8428
Base Version	bver	-1	Number	int	a	RFC 8428
Name	n	0	String	string	a	RFC 8428
Unit	u	1	String	string	a	RFC 8428
Value	v	2	Number	double	a	RFC 8428
String Value	vs	3	String	string	a	RFC 8428
Boolean Value	vb	4	Boolean	boolean	a	RFC 8428
Data Value	vd	8	String	string	a	RFC 8428
Sum	s	5	Number	double	a	RFC 8428
Time	t	6	Number	double	a	RFC 8428
Update Time	ut	7	Number	double	a	RFC 8428

Note that CL = CBOR Label and EI = EXI ID.

Table 7: IANA Registry for SenML Labels

This is the same table as Table 1, with notes removed and columns added for the information that is all the same for this initial set of registrations, but it will need to be supplied with different values for new registrations.

All new entries must define the Name, Label, and XML Type, but the CBOR labels SHOULD be left empty as CBOR will use the string encoding for any new labels. The EI column contains the EXI schemaId value of the first schema that includes this label, or it is empty if this label was not intended for use with EXI. The Reference column SHOULD contain information about where to find out more information about this label.

The JSON, CBOR, and EXI types are derived from the XML type. All XML numeric types such as double, float, integer, and int become a JSON Number. XML boolean and string become a JSON Boolean and String, respectively. CBOR represents numeric values with a CBOR type that does not lose any information from the JSON value. EXI uses the XML types.

New entries can be added to the registration by Expert Review as defined in [RFC8126]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review. New entries should not be made that counteract the advice at the end of Section 4.5.4.

All new SenML Labels that have "base" semantics (see Section 4.1) MUST start with the character "b". Regular labels MUST NOT start with that character. All new SenML Labels with Value semantics (see Section 4.2) MUST have "Value" in their (long-form) name.

Extensions that add a label intended for use with XML need to create a new RelaxNG Schema that includes all the labels in the "SenML Labels" registry.

Extensions that add a label that is intended for use with EXI need to create a new XSD Schema that includes all the labels in the "SenML Labels" registry and then allocate a new EXI schemaId value. Moving to the next letter in the alphabet is the suggested way to create the new value for the EXI schemaId. Any labels with previously blank ID values SHOULD be updated in the "SenML Labels" registry to have their ID set to this new schemaId value.

Extensions that are mandatory to understand to correctly process the Pack MUST have a label name that ends with the "_" character.

12.3. Media Type Registrations

The registrations in the subsections below follow the procedures specified in [RFC6838] and [RFC7303]. This document registers media types for each serialization format of SenML (JSON, CBOR, XML, and EXI) and also a corresponding set of media types for streaming use (SenSML; see Section 4.8). Clipboard formats are defined for the JSON and XML forms of SenML but not for streams or non-textual formats.

The reason there are both SenML and the streaming SenSML formats is that they are not the same data formats, and they require separate negotiation to understand if they are supported and which one is being used. The non-streaming format is required to have some sort of end-of-pack syntax that indicates there will be no more records. Many implementations that receive SenML wait for this end-of-pack marker before processing any of the records. On the other hand, with the streaming formats, it is explicitly not required to wait for this end-of-pack marker. Many implementations that produce streaming SenSML will never send this end-of-pack marker, so implementations that receive streaming SenSML cannot wait for the end-of-pack marker before they start processing the records. Given that SenML and streaming SenML are different data formats, and considering the requirement for separate negotiation, a media type for each one is needed.

12.3.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. See RFC 8428 for details. This simplifies implementation of a very simple system and does not impose any significant limitations as all this data is meant for machine-to-machine communications and is not meant to be human readable.

Security considerations: See Section 13 of RFC 8428.

Interoperability considerations: Applications MUST ignore any JSON key-value pairs that they do not understand unless the key ends with the "_" character, in which case an error MUST be generated. This allows backwards-compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC 8428

Applications that use this media type: The type is used by systems that report, e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+json is supported by using fragment identifiers as specified by RFC 8428.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): senml

Windows Clipboard Name: "JSON Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-json conforms to public.text

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.2. sensml+json Media Type Registration

Type name: application

Subtype name: sensml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. See RFC 8428 for details. This simplifies implementation of a very simple system and does not impose any significant limitations as all this data is meant for machine-to-machine communications and is not meant to be human readable.

Security considerations: See Section 13 of RFC 8428.

Interoperability considerations: Applications MUST ignore any JSON key-value pairs that they do not understand unless the key ends with the "_" character, in which case an error MUST be generated. This allows backwards-compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC 8428

Applications that use this media type: The type is used by systems that report, e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+json is supported by using fragment identifiers as specified by RFC 8428.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): sensml

Macintosh file type code(s): none

Person & email address to contact for further information:
Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.3. senml+cbor Media Type Registration

Type name: application

Subtype name: senml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC 8428 for details.

Security considerations: See Section 13 of RFC 8428.

Interoperability considerations: Applications MUST ignore any key-value pairs that they do not understand unless the key ends with the "_" character, in which case an error MUST be generated. This allows backwards-compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC 8428

Applications that use this media type: The type is used by systems that report, e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+cbor is supported by using fragment identifiers as specified by RFC 8428.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): senmlc

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-cbor conforms to public.data

Person & email address to contact for further information:
Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.4. sensml+cbor Media Type Registration

Type name: application

Subtype name: sensml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC 8428 for details.

Security considerations: See Section 13 of RFC 8428.

Interoperability considerations: Applications MUST ignore any key-value pairs that they do not understand unless the key ends with the "_" character, in which case an error MUST be generated. This allows backwards-compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC 8428

Applications that use this media type: The type is used by systems that report, e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+cbor is supported by using fragment identifiers as specified by RFC 8428.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): sensmlc

Macintosh file type code(s): none

Person & email address to contact for further information:
Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.5. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC 8428 for details.

Security considerations: See Section 13 of RFC 8428.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the "_" character, in which case an error MUST be generated. This allows backwards-compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack.

Published specification: RFC 8428

Applications that use this media type: The type is used by systems that report, e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+xml is supported by using fragment identifiers as specified by RFC 8428.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): senmlx

Windows Clipboard Name: "XML Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-xml conforms to public.xml

Person & email address to contact for further information:
Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.6. sensml+xml Media Type Registration

Type name: application

Subtype name: sensml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC 8428 for details.

Security considerations: See Section 13 of RFC 8428.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the "_" character, in which case an error MUST be generated. This allows backwards-compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack.

Published specification: RFC 8428

Applications that use this media type: The type is used by systems that report, e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+xml is supported by using fragment identifiers as specified by RFC 8428.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): sensmlx

Macintosh file type code(s): none

Person & email address to contact for further information:
Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.7. senml-exi Media Type Registration

Type name: application

Subtype name: senml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC 8428 for details.

Security considerations: See Section 13 of RFC 8428.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the "_" character, in which case an error MUST be generated. This allows backwards-compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack. Further information on using schemas to guide the EXI can be found in RFC 8428.

Published specification: RFC 8428

Applications that use this media type: The type is used by systems that report, e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml-exi is supported by using fragment identifiers as specified by RFC 8428.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): senmle

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-exi conforms to public.data

Person & email address to contact for further information:
Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.8. sensml-exi Media Type Registration

Type name: application

Subtype name: sensml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC 8428 for details.

Security considerations: See Section 13 of RFC 8428.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the "_" character, in which case an error MUST be generated. This allows backwards-compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack. Further information on using schemas to guide the EXI can be found in RFC 8428.

Published specification: RFC 8428

Applications that use this media type: The type is used by systems that report, e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml-exi is supported by using fragment identifiers as specified by RFC 8428.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): sensmle

Macintosh file type code(s): none

Person & email address to contact for further information:
Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.4. XML Namespace Registration

This document registers the following XML namespace in the "IETF XML Registry" defined in [RFC3688].

URI: urn:ietf:params:xml:ns:senml

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces

12.5. CoAP Content-Format Registration

IANA has assigned CoAP Content-Format IDs for the SenML media types in the "CoAP Content-Formats" subregistry within the "Constrained RESTful Environments (CoRE) Parameters" registry [RFC7252]. IDs for the JSON, CBOR, and EXI Content-Formats have been assigned in the 0-255 range (Expert Review), and IDs for the XML Content-Formats have been assigned in the 256-9999 range (IETF Review or IESG Approval). The assigned IDs are shown in the table below:

Media Type	Encoding	ID	Reference
application/senml+json	-	110	RFC 8428
application/sensml+json	-	111	RFC 8428
application/senml+cbor	-	112	RFC 8428
application/sensml+cbor	-	113	RFC 8428
application/senml-exi	-	114	RFC 8428
application/sensml-exi	-	115	RFC 8428
application/senml+xml	-	310	RFC 8428
application/sensml+xml	-	311	RFC 8428

Table 8: CoAP Content-Format IDs

13. Security Considerations

Sensor data presented with SenML can contain a wide array of information that ranges from very public (such as the outside temperature in a given city) to very private (such as patient health information that requires integrity and confidentiality protection). When SenML is used for configuration or actuation, it can be used to change the state of systems and also impact the physical world, e.g., by turning off a heater or opening a lock. Malicious use of SenML to change system state could have severe consequences, potentially including violation of physical security, property damage, and even loss of life.

SenML formats alone do not provide any security and instead rely on the protocol that carries them to provide security. Applications using SenML need to look at the overall context of how these formats will be used to decide if the security is adequate. In particular, for sensitive sensor data and actuation use, it is important to ensure that proper security mechanisms are used to provide, e.g., confidentiality, data integrity, and authentication as appropriate for the usage.

SenML formats defined by this specification do not contain any executable content. However, future extensions could potentially embed application-specific executable content in the data.

SenML Records are intended to be interpreted in the context of any applicable base values. If Records become separated from the Record that establishes the base values, the data will be useless or, worse, wrong. Care needs to be taken in keeping the integrity of a Pack that contains unresolved SenML Records (see Section 4.6).

See also Section 14.

14. Privacy Considerations

Sensor data can range from information with almost no privacy considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transfer protocol such as S/MIME [RFC5751] or HTTP with TLS [RFC2818] that can provide integrity, confidentiality, and authentication information about the source of the data.

The Name fields need to uniquely identify the sources or destinations of the values in a SenML Pack. However, the use of long-term stable and unique identifiers can be problematic for privacy reasons [RFC6973], depending on the application and the potential of these identifiers to be used in correlation with other information. They should be used with care or avoided, for example, as described for IPv6 addresses in [RFC7721].

15. References

15.1. Normative References

[BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th Edition, 2006.

[IEEE.754] IEEE, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754.

[NIST811] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, DOI 10.6028/NIST.SP.811e2008, March 2008.

[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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <https://www.rfc-editor.org/info/rfc7303>.

[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>.

[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>.

[RNC] ISO/IEC, "Information technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG", ISO/IEC 19757-2, Annex C: RELAX NG Compact syntax, December 2008.

[TIME_T] The Open Group Base Specifications, "Open Group Standard - Vol. 1: Base Definitions, Issue 7", Section 4.16, "Seconds Since the Epoch", IEEE Standard 1003.1, 2018, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16>.

[W3C.REC-exi-20140211] Schneider, J., Kamiya, T., Peintner, D., and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)", W3C Recommendation REC-exi-20140211, February 2014, <http://www.w3.org/TR/2014/REC-exi-20140211>.

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

[W3C.REC-xmlschema-1-20041028] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", W3C Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[XPointerElement] Grosso, P., Maler, E., Marsh, J., and N. Walsh, "XPointer element() Scheme", W3C Recommendation REC-xptr-element, March 2003, <https://www.w3.org/TR/2003/REC-xptr-element-20030325/>.

[XPointerFramework]
Grosso, P., Maler, E., Marsh, J., and N. Walsh, "XPointer Framework", W3C Recommendation REC-XPointer-Framework, March 2003, <http://www.w3.org/TR/2003/REC-xptr-framework-20030325/>.

15.2. Informative References

[AN1796] Linke, B., "Overview of 1-Wire Technology and Its Use", Maxim Integrated, Tutorial 1796, June 2008, <http://pdfserv.maximintegrated.com/en/an/AN1796.pdf>.

[CDDL-CBOR]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures", Work in Progress, draft-ietf-cbor-cddl-05, August 2018.

[DEVICE-URN]
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", Work in Progress, draft-ietf-core-dev-urn-02, July 2018.

[IEEE802.1AS]
IEEE, "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks", IEEE Standard 802.1AS.

[IEEE802.1BA]
IEEE, "IEEE Standard for Local and metropolitan area networks--Audio Video Bridging (AVB) Systems", IEEE Standard 802.1BA.

[ISO-80000-5]
ISO, "Quantities and units - Part 5: Thermodynamics", ISO 80000-5, Edition 1.0, May 2007.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", RFC 4151, DOI 10.17487/RFC4151, October 2005, <https://www.rfc-editor.org/info/rfc4151>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[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>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC7111] Hausenblas, M., Wilde, E., and J. Tennison, "URI Fragment Identifiers for the text/csv Media Type", RFC 7111, DOI 10.17487/RFC7111, January 2014, <https://www.rfc-editor.org/info/rfc7111>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[RID-CoRE] Shelby, Z., Vial, M., Groves, C., Zhu, J., and B. Silverajan, Ed., "Reusable Interface Definitions for Constrained RESTful Environments", Work in Progress, draft-ietf-core-interfaces-12, June 2018.

[UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure", Version 2.1, Regenstrief Institute and the UCUM Organization, November 2017, <http://unitsofmeasure.org/ucum.html>.

Acknowledgements

We would like to thank Alexander Pelov, Alexey Melnikov, Andrew McClure, Andrew McGregor, Bjoern Hoehrmann, Christian Amsuess, Christian Groves, Daniel Peintner, Jan-Piet Mens, Jim Schaad, Joe Hildebrand, John Klensin, Karl Palsson, Lennart Duhrsen, Lisa Dusseault, Lyndsay Campbell, Martin Thomson, Michael Koster, Peter Saint-Andre, Roni Even, and Stephen Farrell, for their review comments.

Authors' Addresses

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Email: fluffy@iii.ca

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
United States of America

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

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

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