Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 7, 2016

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson

April 5, 2016

Media Types for Sensor Markup Language (SenML)
draft-jennings-core-senml-06

Abstract

This specification defines media types for representing simple sensor measurements and device parameters in the Sensor Markup Language (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 this media type in protocols such as HTTP or 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 new media types for carrying simple sensor information in a protocol such as HTTP or CoAP called the Sensor Markup Language (SenML). This format was 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 relatively efficiently collect a large number of sensor measurements. The markup language can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor 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 reason. Other information about the sensor can be discovered by other methods such as using the CoRE Link Format [RFC6690].

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single array that contains a series of SenML Records which can each contain attributes such as an 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 [RFC7159], CBOR [RFC7049], XML, and Efficient XML Interchange (EXI) [W3C.REC-exi-20110310].

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

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

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 23.5 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets on mesh networks from large numbers of constrained devices. Keeping the total size of payload under 80 bytes makes this easy to use on a wireless mesh network. 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 a 8 bit microprocessor. Experience with Google power meter and 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 in the "v" tag, the time of a measurement is in the "t" tag, the "n" tag has a unique sensor name, and the unit of the measurement is carried in the "u" tag.

```
[
    { "n": "urn:dev:ow:10e2073a01080063",
      "t": 1276020076, "v":23.5, "u":"Cel" },
    { "n": "urn:dev:ow:10e2073a01080063",
      "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

To keep the messages small, it does not make sense to repeat the "n" tag 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",
      "t": 1276020076, "v":23.5, "u":"Cel" },
    {  "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

In the above example the Base Name is in the "bn" tag and the "n" tags in each Record are the empty string so they are omitted. The Base Name also could be put in a separate Record such as in the following example.

```
[
    { "bn": "urn:dev:ow:10e2073a01080063" },
    { "t": 1276020076, "v":23.5, "u":"Cel" },
    { "t": 1276020091, "v":23.6, "u":"Cel" }
]
```

Some devices have accurate time while others do not so SenML supports absolute and relative times. Time is represented in floating point as seconds and values greater than zero represent an absolute time relative to the unix epoch while values of 0 or less represent a relative time in the past from the current time. A simple sensor with no absolute wall clock time might take a measurement every second and batch up 60 of them then send it to a server. It would include the relative time the measurement was made to the time the batch was send in the SenML Pack. The server might have accurate NTP time and 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 Pack 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 [RFC2119].

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.

4. Semantics

Each SenML Pack carries a single array that represents a set of measurements and/or parameters. This array contains a series of objects with several optional attributes described below:

Base Name: This is a string that is prepended to the names found in the entries. This attribute is optional. This applies to the entries in all Records. A Base Name can only be included in the first Record of the array.

Base Time: A base time that is added to the time found in an entry. This attribute is optional. This applies to the entries in all Records. A Base Time can only be included in the first Record of the array.

Base Unit: A base unit that is assumed for all entries, unless otherwise indicated. This attribute is optional. If a record does not contain a unit value, then the base unit is used otherwise the value of found in the Unit is used. This applies to the entries in all Records. A Base Unit can only be included in the first object of the array.

Base Value: A base value is added to the value found in an entry, similar to Base Time. This attribute is optional. This applies to the entries in all Records. A Base Value can only be included in the first Record of the array.

Version: Version number of media type format. This attribute is optional positive integer and defaults to 5 if not present. A Version can only be included in the first object of the array.

Name: Name of the sensor or parameter. When appended to the Base Name attribute, 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, 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: Units for a measurement value. Optional. If the Record has not Unit, the Base Unit is used as the Unit. Having no Unit and no Base Unit is allowed.

Value Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using three basic data types, Floating point numbers ("v" field for "Value"), Booleans ("vb" for "Boolean Value"), Strings ("vs" for "String Value") and Data ("vd" for "Binary Data Value") . Exactly one of these three fields MUST appear unless there is Sum field in which case it is allowed to have no Value field or to have "v" field.

Sum: Integrated sum of the values over time. Optional. This attribute is in the units specified in the Unit value multiplied by seconds.

Time: Time when value was recorded. Optional.

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

The SenML format can be extended with further custom attributes. TODO - describe what extensions are possible and how to do them.

Systems reading one of the objects MUST check for the Version attribute. If this value is a version number larger than the version which the system understands, the system SHOULD NOT use this object. This allows the version number to indicate that the object contains mandatory to understand attributes. New version numbers can only be defined in an RFC that updates this specification or it successors.

The Name value is concatenated to the Base Name value to get the name of the sensor. The resulting name needs to uniquely identify and differentiate the sensor from all others. If the object is a representation resulting from the request of a URI [RFC3986], then in the absence of the Base Name attribute, this URI is used as the default value of Base Name. Thus in this case the Name field needs to be unique for that URI, for example an index or subresource name of sensors handled by the URI.

Alternatively, for objects not related to a URI, a unique name is required. In any case, it is RECOMMENDED that the full names are represented as URIs or URNs [RFC2141]. One way to create a unique name is to include some bit string that has guaranteed uniqueness (such as a 1-wire address) that is assigned to the device. Some of the examples in this draft use the device URN type as specified in [I-D.arkko-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name. Note that long-term stable unique identifiers are problematic for privacy reasons [RFC7721] and should be used with care or avoided.

The resulting concatenated name MUST consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", or "_" and 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 these names can be directly used as in other types of URI including segments of an HTTP path with no special encoding and can be directly used in many databases and analytic systems. [RFC5952] contains advice on encoding an IPv6 address in a name.

If either the Base Time or Time value is missing, the missing attribute is considered to have a value of zero. The Base Time and Time values are added together to get the time of measurement. A time of zero indicates that the sensor does not know the absolute time and the measurement was made roughly "now". A negative value is used to indicate seconds in the past from roughly "now". A positive value is used to indicate the number of seconds, excluding leap seconds, since the start of the year 1970 in UTC.

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

5. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry significant static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML Packs, this meta-data 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=) attribute.

6. JSON Representation (application/senml+json)

Record atributes:

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

The root content consists of an array with JSON objects for each SenML Record. All the fields in the above table MAY occur in the records with the type specified in the table.

Only the UTF-8 form of JSON is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC4627]. Characters in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648].

Systems receiving measurements MUST be able to process the range of floating point numbers that are representable as an IEEE double-precision floating-point numbers [IEEE.754.1985]. 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. The mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than 5 characters long. This allows time values to have better than micro second precision over the next 100 years.

6.1. Examples

TODO - simplify examples

TODO - Examples are messed up on if time is an integer or float

TODO - Add example with string, data, boolean, and base value

6.1.1. Single Datapoint

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", "v":23.1, "u":"Cel" }]
```

6.1.2. Multiple Datapoints

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

```
[{"bn": "urn:dev:ow:10e2073a01080063/"},
 { "n": "voltage", "t": 0, "u": "V", "v": 120.1 },
 { "n": "current", "t": 0, "u": "A", "v": 1.2 }
]
```

The next example is similar to the above one, but shows current at Tue Jun 8 18:01:16 UTC 2010 and at each second for the previous 5 seconds.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
  "bt": 1276020076.001,
  "bu": "A",
  "bver": 5},
   { "n": "voltage", "u": "V", "v": 120.1 },
   { "n": "current", "t": -5, "v": 1.2 },
   { "n": "current", "t": -4, "v": 1.30 },
   { "n": "current", "t": -3, "v": 0.14e1 },
   { "n": "current", "t": -2, "v": 1.5 },
   { "n": "current", "t": -1, "v": 1.6 },
   { "n": "current", "t": 0,  "v": 1.7 }
]
```

Note that in some usage scenarios of SenML the implementations MAY 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 a separate mime type (TODO) to indicate Sensor Streaming Markup Language (SensML) for this usage. 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 not have to wait for the full SensML Stream to be complete.

For instance, the following stream of measurements may be sent via a long lived HTTP POST from the producer of a SensML to the consumer of that, and each measurement object may be reported at the time it measured:

```
[ {"bn": "urn:dev:ow:10e2073a01080063",
  "bt": 1320067464,
  "bu": "%RH"},
   { "v": 21.2, "t": 0 },
   { "v": 21.3, "t": 10 },
   { "v": 21.4, "t": 20 },
   { "v": 21.4, "t": 30 },
   { "v": 21.5, "t": 40 },
   { "v": 21.5, "t": 50 },
   { "v": 21.5, "t": 60 },
   { "v": 21.6, "t": 70 },
   { "v": 21.7, "t": 80 },
   { "v": 21.5, "t": 90 },
...
```

6.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with an IPv6 address 2001:db8::1, 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 time is used to for correlating 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": 1320067464,
  "bu": "%RH"},
   { "v": 20.0, "t": 0 },
   { "v": 24.30621, "u": "lon", "t": 0 },
   { "v": 60.07965, "u": "lat", "t": 0 },
   { "v": 20.3, "t": 60 },
   { "v": 24.30622, "u": "lon", "t": 60 },
   { "v": 60.07965, "u": "lat", "t": 60 },
   { "v": 20.7, "t": 120 },
   { "v": 24.30623, "u": "lon", "t": 120 },
   { "v": 60.07966, "u": "lat", "t": 120 },
   { "v": 98.0, "u": "%EL", "t": 150 },
   { "v": 21.2, "t": 180 },
   { "v": 24.30628, "u": "lon", "t": 180 },
   { "v": 60.07967, "u": "lat", "t": 180 }
]
```

The size of this example represented in various forms, as well as that form compressed with gzip is given in the following table.

Encoding	Size	Compressed Size
JSON	567	200
XML	656	232
CBOR	292	192
EXI	160	183

Table 1: Size Comparisons

Note the CBOR and EXI sizes are not using the schema guidance so the could be a bit smaller.

6.1.4. Collection of Resources

The following example shows how to query one device that can provide multiple measurements. 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.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
  "bt": 1320078429,
  "bver": 5
  },
  { "n": "temperature", "v": 27.2, "u": "Cel" },
  { "n": "humidity", "v": 80, "u": "%RH" }
]
```

7. CBOR Representation (application/senml+cbor)

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

o For compactness, the CBOR representation uses integers for the map keys defined in Table 2. This table is conclusive, i.e., there is no intention to define any additional integer map keys; any extensions will use string map keys.

o For JSON Numbers, the CBOR representation can use integers, floating point numbers, or decimal fractions (CBOR Tag 4); the common limitations of JSON implementations are not relevant for these. For the version number, however, only an unsigned integer is allowed.

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

Table 2: CBOR representation: integers for map keys

The following example shows an hexdump of the CBOR example for the same sensor measurement as in Section 6.1.2.

```
0000 88 a4 62 62 6e 78 1c 75 72 6e 3a 64 65 76 3a 6f |..bbnx.urn:dev:o|
0010 77 3a 31 30 65 32 30 37 33 61 30 31 30 38 30 30 |w:10e2073a010800|
0020 36 33 2f 62 62 74 1a 4c 0e 85 6c 62 62 75 61 41 |63/bbt.L..lbbuaA|
0030 63 76 65 72 05 a3 61 6e 67 76 6f 6c 74 61 67 65 |cver..angvoltage|
0040 61 75 61 56 61 76 fb 40 5e 06 66 66 66 66 66 a3 |auaVav.@^.fffff.|
0050 61 6e 67 63 75 72 72 65 6e 74 61 74 24 61 76 fb |angcurrentat$av.|
0060 3f f3 33 33 33 33 33 33 a3 61 6e 67 63 75 72 72 |?.333333.angcurr|
0070 65 6e 74 61 74 23 61 76 fb 3f f4 cc cc cc cc cc |entat#av.?......|
0080 cd a3 61 6e 67 63 75 72 72 65 6e 74 61 74 22 61 |..angcurrentat"a|
0090 76 fb 3f f6 66 66 66 66 66 66 a3 61 6e 67 63 75 |v.?.ffffff.angcu|
00a0 72 72 65 6e 74 61 74 21 61 76 fb 3f f8 00 00 00 |rrentat!av.?....|
00b0 00 00 00 a3 61 6e 67 63 75 72 72 65 6e 74 61 74 |....angcurrentat|
00c0 20 61 76 fb 3f f9 99 99 99 99 99 9a a2 61 6e 67 | av.?........ang|
00d0 63 75 72 72 65 6e 74 61 76 fb 3f fb 33 33 33 33 |currentav.?.3333|
00e0 33 33 0a                                        |33.|
00e3
```

8. XML Representation (application/senml+xml)

A SenML Stream can also be represented in XML format as defined in this section. The following example shows an XML example for the same sensor measurement as in Section 6.1.2.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a01080063/" bt="1.276020076e+09"
  bu="A" bver="5"></senml>
  <senml 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 tag that contains a series of senml tags for each SenML Record. The SenML Fields are represents as XML attributes. The following table shows the mapping the SenML Field names to the attribute used in the XML senml tag.

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

The RelaxNG schema for the XML is:

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

   link = element l {
         attribute * { xsd:string }*
   }


   senml = element senml {
     attribute bn { xsd:string }?,
     attribute bt { xsd:double }?,
     attribute bv { 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 }?,

     link*
   }

   sensml =
     element sensml {
       senml+
   }

   start = sensml
```

9. EXI Representation (application/senml-exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema 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. This mode however does not allow tag extensions to the schema, and 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 option MUST be included. An EXI schemaID options MUST be set to the value of "a" indicating the scheme provided in this specification. Future revisions to the schema can change this 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.

TODO - examples probably have the wrong setting the schemaID

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="l">
    <xs:complexType>
      <xs:anyAttribute processContents="skip" />
    </xs:complexType>
  </xs:element>
  <xs:element name="senml">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded"
        ref="ns1:l" />
      </xs:sequence>
      <xs:attribute name="bn" type="xs:string" />
      <xs:attribute name="bt" type="xs:double" />
      <xs:attribute name="bv" 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 this example is the same information as the first example in Section 6.1.2 in JSON format.

```
   <sensml xmlns="urn:ietf:params:xml:ns:senml">
     <senml bn="urn:dev:ow:10e2073a01080063/"></senml>
     <senml 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 41 cd 95 b9 b5 b0 d4 b9 9d 95 b8 b9 e1 cd |.0A.............|
 0010 91 00 f3 ab 93 71 d3 23 2b b1 d3 7b b9 d1 89 83 |.....q.#+..{....|
 0020 29 91 81 b9 9b 09 81 89 81 c1 81 81 b1 99 7f 14 |)...............|
 0030 25 d9 bd b1 d1 85 9d 94 80 d5 8a c4 26 01 0a 12 |%...........&...|
 0040 c6 ea e4 e4 ca dc e8 40 68 24 19 00 90          |.......@h$...|
 004d
```

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 makes 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 82 0e 6c ad cd ad 86 a5 cc ec ad c5 cf |..H..l..........|
 0010 0e 6c 80 02 05 1d 75 72 6e 3a 64 65 76 3a 6f 77 |.l....urn:dev:ow|
 0020 3a 31 30 65 32 30 37 33 61 30 31 30 38 30 30 36 |:10e2073a0108006|
 0030 33 02 05 43 65 6c 01 00 e7 01 01 00 04 01       |3..Cel........|
 003e
```

A small temperature sensor devices that only generates this one EXI file does not really need an full EXI implementation. It can simple hard code the output replacing the one wire device ID starting at byte 0x16 and going to byte 0x31 with it’s device ID, and replacing the value "0xe7 0x01" at location 0x38 to 0x39 with the current temperature. The EXI Specification [W3C.REC-exi-20110310] 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 bytes at is set to 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.

10. Usage Considerations

The measurements support sending both the current value of a sensor as well as the an integrated sum. For many types of measurements, the sum is more useful than the current value. 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 units set to watts, but it would put the sum of energy used in the "s" attribute of the measurement. It might optionally include the current power in the "v" attribute.

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. Implementors 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 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 always positive, the sum should never get smaller; so if the sum does get smaller, the application will know that one of the situations listed above has happened.

11. CDDL

For reference, the CBOR representation can be described with the CDDL [I-D.greevenbosch-appsawg-cbor-cddl] specification in Figure 1.

```
SenML-Pack = [initial-record, * follow-on-record]

initial-record = initial-defined .and initial-generic
follow-on-record = follow-on-defined .and follow-on-generic

; first do a specification of the labels as defined:

initial-defined = {
  ? bn => tstr,        ; Base Name
  ? bt => numeric,     ; Base Time
  ? bu => tstr,        ; Base Units
  ? bv => numeric,     ; Base value
  ? bver => uint,      ; Base Version
  follow-on-defined-group,
  + base-key-value-pair
}

follow-on-defined-group = (
   ? n => tstr,        ; Name
   ? u => tstr,        ; Units
   ? ( v => numeric // ; Numeric Value
       vs => tstr //   ; String Value
       vb => bool //   ; Boolean Value
       vd => bstr )    ; Data Value
   ? s => numeric,     ; Value Sum
   ? t => numeric,     ; Time
   ? ut => numeric,    ; Update Time
   * key-value-pair
)
follow-on-defined = { follow-on-defined-group }

; CBOR version (use the labels)
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
; use the label *names* for JSON

; now define the generic versions
```

```
   initial-generic = {
     follow-on-generic-group,
     * base-key-value-pair,
   }

   follow-on-generic-group = (
     + key-value-pair,
   )
   follow-on-generic = { follow-on-generic-group }

   key-value-pair = ( non-b-label => value )

   base-key-value-pair = ( b-label => value )

   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 / bstr / numeric / bool
   numeric = number / decfrac
```

Figure 1: CDDL specification for CBOR SenML

12. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification.

12.1. Units Registry

IANA will create a registry of SenML unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in location such as [NIST811] and [BIPM].

Symbol	Description	Type	Reference
m	meter	float	RFC-AAAA
g	gram	float	RFC-AAAA
s	second	float	RFC-AAAA
A	ampere	float	RFC-AAAA
K	kelvin	float	RFC-AAAA
cd	candela	float	RFC-AAAA
mol	mole	float	RFC-AAAA
Hz	hertz	float	RFC-AAAA
rad	radian	float	RFC-AAAA
sr	steradian	float	RFC-AAAA

N	newton	float	RFC-AAAA
Pa	pascal	float	RFC-AAAA
J	joule	float	RFC-AAAA
W	watt	float	RFC-AAAA
C	coulomb	float	RFC-AAAA
V	volt	float	RFC-AAAA
F	farad	float	RFC-AAAA
Ohm	ohm	float	RFC-AAAA
S	siemens	float	RFC-AAAA
Wb	weber	float	RFC-AAAA
T	tesla	float	RFC-AAAA
H	henry	float	RFC-AAAA
Cel	degrees Celsius	float	RFC-AAAA
lm	lumen	float	RFC-AAAA
lx	lux	float	RFC-AAAA
Bq	becquerel	float	RFC-AAAA
Gy	gray	float	RFC-AAAA
Sv	sievert	float	RFC-AAAA
kat	katal	float	RFC-AAAA
pH	pH acidity	float	RFC-AAAA
%	Value of a switch (note 1)	float	RFC-AAAA
count	counter value	float	RFC-AAAA
%RH	Relative Humidity	float	RFC-AAAA
m2	area	float	RFC-AAAA
l	volume in liters	float	RFC-AAAA
m/s	velocity	float	RFC-AAAA
m/s2	acceleration	float	RFC-AAAA
l/s	flow rate in liters per second	float	RFC-AAAA
W/m2	irradiance	float	RFC-AAAA
cd/m2	luminance	float	RFC-AAAA
Bspl	bel sound pressure level	float	RFC-AAAA
bit/s	bits per second	float	RFC-AAAA
lat	degrees latitude (note 2)	float	RFC-AAAA
lon	degrees longitude (note 2)	float	RFC-AAAA
%EL	remaining battery energy level in percents	float	RFC-AAAA
EL	remaining battery energy level in seconds	float	RFC-AAAA
beat/m	Heart rate in beats per minute	float	RFC-AAAA
beats	Cumulative number of heart beats	float	RFC-AAAA

Table 3

- Note 1: A value of 0.0 indicates the switch is off while 1.0 indicates on and 0.5 would be half on.

- Note 2: Assumed to be in WGS84 unless another reference frame is known for the sensor.

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. 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. Units should define the semantic information and be chosen carefully. Implementors 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 be 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 SI prefixes such as "k" before the unit is not allowed. Instead one can represent the value using scientific notation such a 1.2e3. TODO - Open Issue. Some people would like to have SI prefixes to improve human readability.

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

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 the are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, 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, ph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.

9. The unit names are case sensitive and the correct case needs to be used, but 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 the / indicates that the units that follow are the reciprocal. 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 label registry

IANA will create a registry for SenML labels. The initial content of the registry are shown in TODO.

New entries can be added to the registration by either Expert Review or IESG Approval as defined in [RFC5226]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review.

12.3. Media Type Registration

The following registrations are done following the procedure specified in [RFC6838] and [RFC7303].

12.3.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json and sensml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7159]. See RFC-AAAA for details. This simplifies implementation of 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: Sensor data can contain a wide range of information ranging from information that is very public, such the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. This format does not provide any security and instead relies on the transport protocol that carries it to provide security. Given applications need to look at the overall context of how this media type will be used to decide if the security is adequate.

Interoperability considerations: Applications should ignore any JSON key value pairs that they do not understand. This allows backwards compatibility extensions to this specification. The "ver" 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-AAAA

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

Additional information:

Magic number(s): none

File extension(s): senml

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

12.3.2. senml+cbor Media Type Registration

Type name: application

Subtype name: senml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

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

Additional information:

Magic number(s): none

File extension(s): senml

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+xml Media Type Registration

Type name: application

Subtype name: senml+xml and sensml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

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

12.3.4. senml-exi Media Type Registration

Type name: application

Subtype name: senml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

Published specification: RFC-AAAA

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): senml

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 namespaces 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 is requested to assign CoAP Content-Format IDs for the SenML media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. All IDs are assigned from the "Expert Review" (0-255) range. The assigned IDs are show in Table 4.

Media type	ID
application/senml+json	TBD
application/sensml+json	TBD
application/senml+cbor	TBD
application/senml+xml	TBD
application/sensml+xml	TBD
application/senml-exi	TBD

Table 4: CoAP Content-Format IDs

13. Security Considerations

See Section 14. Further discussion of security properties can be found in Section 12.3.

14. Privacy Considerations

Sensor data can range from information with almost no security 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 transport protocol such as S/MIME or HTTP with TLS that can provide integrity, confidentiality, and authentication information about the source of the data.

15. Acknowledgement

We would like to thank Lisa Dusseault, Joe Hildebrand, Lyndsay Campbell, Martin Thomson, John Klensin, Bjoern Hoehrmann, Carsten Bormann, and Christian Amsuess for their review comments.

The CBOR Representation text and CDDL was contributed by Carsten Bormann.

16. References

16.1. Normative References

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

[IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754, August 1985.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <http://www.rfc-editor.org/info/rfc4627>.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://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, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

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

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

[W3C.REC-exi-20110310]
Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <http://www.w3.org/TR/2011/REC-exi-20110310>.

16.2. Informative References

[I-D.arkko-core-dev-urn]
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-03 (work in progress), July 2012.

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-08 (work in progress), March 2016.

[I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-04 (work in progress), November 2015.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, DOI 10.17487/RFC2141, May 1997, <http://www.rfc-editor.org/info/rfc2141>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc4122>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

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

[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, <http://www.rfc-editor.org/info/rfc7721>.

[UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics, 2013, <http://unitsofmeasure.org/ucum.html>.

Appendix A. Links extension

An extension to SenML to support links is expected to be registered and defined by [I-D.ietf-core-links-json].

The link extension can be an array of objects that can be used for additional information. Each object in the Link array is constrained to being a map of strings to strings with unique keys.

The following shows an example of the links extension.

```
[{"bn": "urn:dev:ow:10e2073a01080063/",
  "bt": 1320078429,
  "l": "[{\"href\":\"humidity\",\"foo\":\"bar1\"}\"
  },
  { "n": "temperature", "v": 27.2, "u": "Cel" },
  { "n": "humidity", "v": 80, "u": "%RH" }
]
```

Authors' Addresses

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

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

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