Data serialization / value formats

What is it and why?

The VSS Taxonomy defines what data "entities" (Signals and Attributes) we can deal with, and are used in the protocol(s) defined by W3C Automotive Working group, as well as other initiatives inside and outside the vehicle.

But in addition to VSS itself, we need to define the data-exchange formats for ***measured values*** of those Signals. This starts by defining terms, but quickly develops into defining one or several variants of the actual message content format, whether in JSON or other.

Relationship to Protocols

Data formats sometimes overlap protocol definitions because some protocols (but not all) define the data format in its specification. VISS / W3C Gen2 is an example of a protocol definition that defines both the protocol interactions between client and server, and the data exchange format that fits the VSS model. Ultimately, any chosen (stack of) protocols must at some point define the transferred data formats, otherwise no understandable exchange can be had, and this page is intended to support the development of such a definition.

Looking at a wider set of protocols it is clear that we have some more work remaining.

The data exchange protocols we discuss fall into **different categories**, each requiring some more work on defining value exchange data formats:

1. **A protocol does not (yet) cover all variations of data exchange.**
    a. When this page was written, W3C VISS protocol (v2) supported subscription to updates and on-demand fetching of the current value of one or several, specified signals in one go. In its latest form it has a significant number of query parameters and filters, and supports the fetching of a series of historical recorded values (i.e. TimeSeries according to the definitions below). VISS v2 now specifies features that mirror most of the types of messages listed here, with the exception of Snapshots.
2. **A protocol defines only a "transport"**
    a. We often discuss protocols that define some behavior of data transfer, such as pub/sub semantics, but they are designed to be generic and therefore support *any* type of information to be transferred by the protocol. This means they do not (can not) define the format of the content of the data container (payload). Such transport protocols are set up to transfer any arbitrary sequence of bytes. This makes those technologies widely applicable, but selecting them is not enough without also defining the payload format. Examples of some such protocols would be MQTT or WAMP, but the principle extends to *many* generic protocols and frameworks.
3. **A protocol defines transport, query semantics, and even a few expectations for the exchanged data format, but is still generic and requires additional definitions to become unambiguous for a particular case.**
    a. Example: GraphQL is a generic technology that clarifies a bit more about expected data semantics and formats but it still requires a schema to be defined to indicate the exact underlying data model, what types of queries can be made using the GraphQL language, and other details such as the datatypes that are expected to be returned. A schema must be defined for GraphQL, and for other similar situations, and that schema might also be derived from this generic analysis.
    b. Example 2: To consume and process data in Apache Spark, Kafka and presumably for many other generic data-handling frameworks we also need to define schemas that define the format and content of the transferred data, in a similar fashion. These protocols might also match category 2/3.

Related references

- MF4/MD4 data format
    - A C++ implementation

Definitions

(proposal, open for discussion)

Signal:

- An observable single piece of data in a Data Taxonomy, defined by its unique identifier
- In practice we here mean the absolute path of a leaf-node in a VSS taxonomy

Request:

- A request to deliver a measurement or measurements, as according to the chosen communication protocol.
- For example an instance of GET in VISS/W3C Gen2, or a Query in GraphQL.
- This is assumed to request a set of data that has already been measured, (or if it is an instantaneous value, a value that is measured on-demand and delivered instantly).

Job:

- A request to make measurements, typically some time in the future.

- Unlike Request, which asks for **Data Delivery**, a Job definition is used to instruct a system to perform a measurement or several measurements over time, at some time in the future.
- A Job does not deliver data until it is requested.
- A Job may include conditions such as a time-period, but in advanced systems also other logical conditions that should be fulfilled for the job to execute.
- TODO: Compare SENSORIS work on this.

⚠ This page is currently concerned with the payload, and not a full protocol definition, so no further definition of **Request** or **Job** is made here.

Observation:

- The act of making a measurement, a.k.a. to record one or several *values* for a particular named data item.

Data Package:

= A delivery of data sent at a particular time.

(think of it as the whole Message that is in response to a Request)

This will likely need to include some metadata regarding the request:

- **Vehicle identity** – should this be special or is just a measurement on a VSS attribute. For example VIN number is already defined in VSS. Presumably it could be just a VSS defined data item?
  - Depending on the specific implementation of this concept this might be an "anonymous" ID instead of one that can identify a vehicle or person.
- **Job ID** (when applicable)
- Sequence number (if partial delivery of a Job)
- The values container itself (type **Snapshot**, **Bundle**, or single **Record**)

Additional Metadata

Following input given in the W3C data TF:

- Observation metadata
  - Sampling/Compression methods
  - Transmission method
  - Consent requirements
  - Retention time
- Signal metadata
  - Sensitivity
  - Quality
  - Unit

Record:

- A container to represent one single measured data value
- Subtypes **(Record Types)** indicate which Signal has been measured.
- Records are used to represent data with associated metadata, which are different depending on the Record Type, as needed for different cases
- Example of possible record types:
  - Just the value.
  - The value plus a timestamp
  - The value plus a timestamp plus a timestamp accuracy information
  - The value plus additional qualitative information
  - The value plus a timestamp plus the location (e.g. GNSS position) where a measurement was performed.
- All of the above may also specify the signal name, or not: Some record types may need to specify the signal it is referring to, but others might not, because the record is delivered in a context where it is known which signal is being measured.

❓ *Why not just use a single record type (superset of all functionality)?*

A: The reason would be to optimize the performance and bandwidth. In other words, don't transfer what is not needed for a certain case. If a timestamp is not needed, we should make sure we support transferring data without providing a timestamp, for example. Hence, this proposes a simple class hierarchy of sub-types of Record.

Record Subtypes:

- ~~**PlainRecord**~~ there is no need to differentiate it from Record unless we want the top parent type (Record) to be "abstract" and only allow subtypes be concrete types.
  As far as I can see there is nothing to gain from that. So we can consider Record to be a concrete parent type and **Record == Plain Record**

- **TimeStampedRecord**
- **ToleranceTimeStampedRecord,** ... ?

+ Record types which specify the signal name inside:

- **SpecifiedRecord**
- **SpecifiedTimeStampedRecord**, etc.

Record types which specify the geospatial position in addition to the time value:

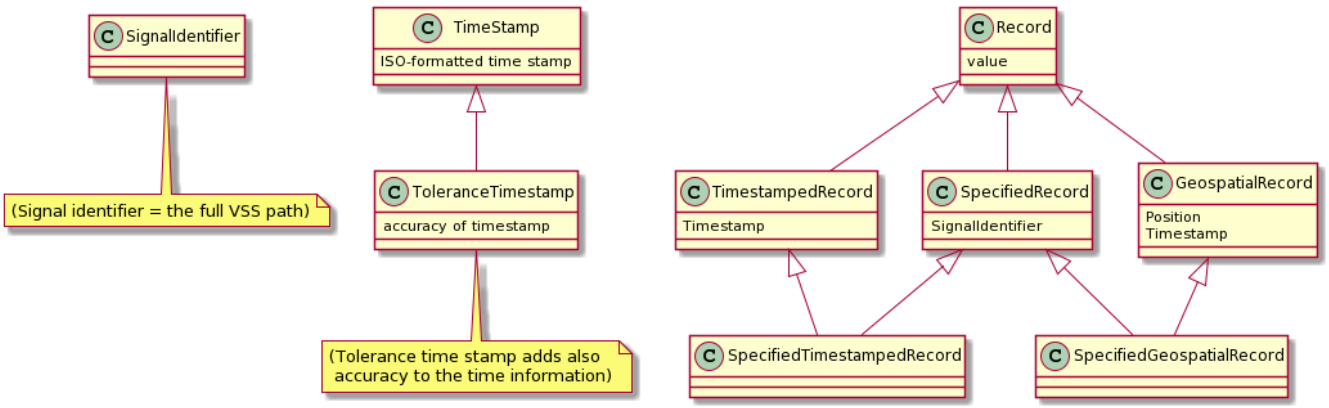
- **GeospatialRecord**
- **SpecifiedGeospatialRecord**

(N.B. In this proposal, Geospatial records always include a time stamp, because it seems to be the overwhelmingly dominant usage, but variations without time stamp would of course be possible)

DerivedRecord and StatisticsRecord

- This is a record type that does not deliver the original data but something that is calculated from it.
- These are needed only if the "derived" signal is not already listed in the VSS signal database. In other words, it is perfectly possible to use VSS to define some derived or statistical value already, by giving it a name and definition in the data catalog/schema. A normal request for that VSS signal would then deliver the value in a normal record, and there is no need to define in the value-measurement protocols how this value is calculated since it is declared in the VSS description or simply decided by the system that delivers it.
- Here is one example of this already listed in VSS today: **Vehicle.AverageSpeed.** It has the description "Average speed for the current trip". You could imagine defining another signal named **Vehicle.Speed.MonthlyAverage,** for example and just fetch it as a normal Record.
- ... but if the VSS catalog does not give a name to the derived or statistics calculation, then this can be done by defining new message types.
- A VSS signal has a single description so its meaning is well defined and not changing. A **DerivedRecord** type such as statistical record complements this since it can have modifiable parameters, such as over which time period is the measurement done.
- Subtypes of DerivedRecord are **StatisticsRecord** and maybe some others (e.g. mathematical function / curve matching?)
- Further Subtypes of **StatisticsRecord**: **Average**, **Median**, **Max**, **Min**, **Histogram**, and so on.
- (Additional refinement possible here)

Overview



TimeStamp

NOTE: The exact format of time stamps (and any other data representation) may differ when these concepts are translated to different protocols or languages, as long as the original meaning as required by the VSS specification remains.

1. Text Format

One option is to use a string and it is then recommended to use the ISO 8601 standard format, with fractional seconds (e.g. microseconds) and always UTC (Zulu) time zone.

Real, "Wall clock time"

- "ts" : "2020-01-10T02:59:43.492750Z # Zulu time, ISO std with microseconds

Relative to a previously predefined time stamp reference:

- "rts" : "T02:59:43.100044" # Similar to ISO 8610. Years/month/dates can be omitted if zero
- "rts" : "02:59:43.100044" # Alternative, also OK
- "rts2", "rts3", ... # If more than one relative time stamp reference had been previously agreed

Binary format

For some purposes more efficient binary encodings should be considered, such as an integer of appropriate size, usually containing fractions of seconds relative to a known starting point.

Bundle

- A collection of several records, transferred together.
- On this page, two subtypes are defined: **TimeSeries** and **Snapshot**.

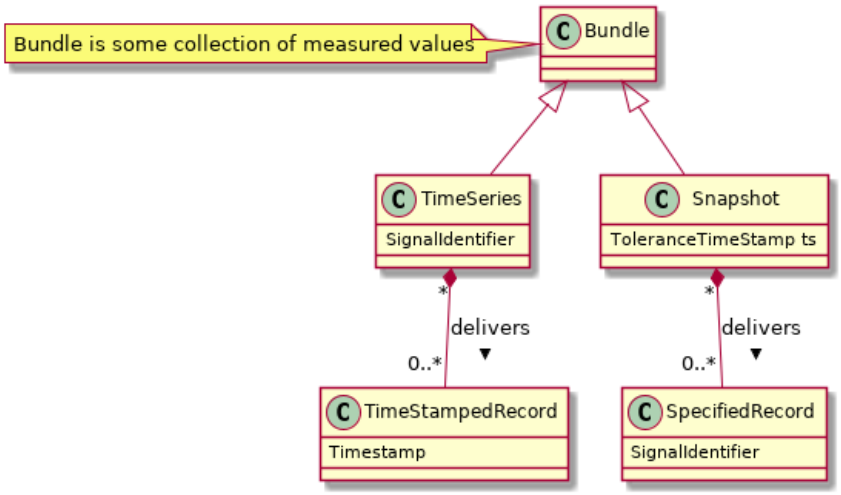
TimeSeries

- Several measured values, **of the same signal**, taken over a period of time.
- A TimeSeries contains time stamp for each value
- A TimeSeries is a collection of Records

Snapshot:

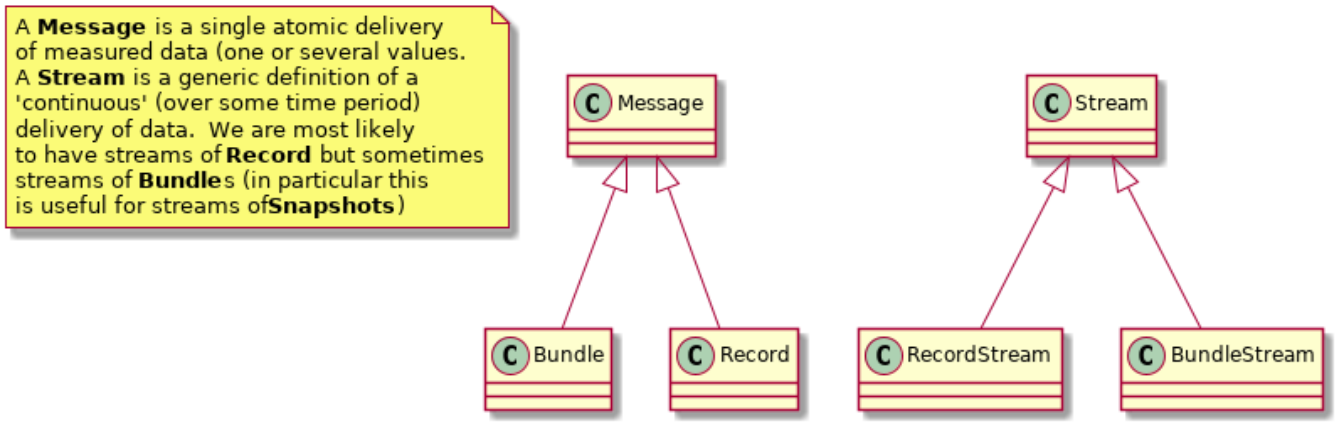
- Several measured values, **of different signals**, that have a relationship to each other because they were measured "at the same time".
- ⚠️ Due to potential time sync limitations or timestamp inaccuracy, this concept of "at the same time" could be defined as a time range of a chosen length)
- A snapshot is built up of several Records, and additional information
- A **Job** might define beforehand, which different signals shall be grouped into the Snapshot instances.
- A **Snapshot** can be seen as a generalisation of the **"Freeze Frame"** concept used in automotive diagnostics.
    - Side note: Freeze Frames are sometimes delivered as an opaque data dump that can only be interpreted by those who know the internal structure (and this binary-blob could then be transferred within a single VSS signal definition) but we are here proposing an understandable and readable format for **Snapshot.** It is implied by the example, that this is achieved because the snapshot only contains values from signals that exist in the VSS catalog/schema of the system, each identified by VSS path)
- In the proposed example, one Snapshot is defined to only have one measurement per signal.

Note that values in a Snapshot need a record type that specifies the signal, i.e. (a subtype of) **SpecifiedRecord,** since different signals are included in the same message.



Stream:

- Continuous delivery of data points according to a predefined agreement
- This is not to be seen as a different container type. It is more a definition of the delivery method (constant stream compared to atomic message)
- A stream does not have a fixed start/end time
- It delivers a stream of **Records** (or Bundles, although Snapshot is the most likely subtype. A stream of TimeSeries is likely redundant, and might often just be a stream of Records instead)
    - It might also deliver side-band information (e.g. Job information)
- It is related to a delivery of a **Subscription** for protocols that support subscriptions.



Examples, using JSON

(Plain) **Record**:

```
{
   "value" : " 100.54"
}
```

TimestampedRecord:

```
{
   "ts" : "2020-01-10T02:59:43.491751Z   # Zulu time, ISO std with microseconds
   "value" : "42"
}
```

GeospatialRecord:

```
{
   "pos" : "[format tbd]"
   "ts" : "2020-01-10T02:59:43.491751Z   # Zulu time, ISO std with microseconds
   "value" : "42"
}
```

SpecifiedRecord:

```
{
   "signal" : "vehicle.Chassis.Axle2.WheelCount"
   "value" : "2"
}
```

SpecifiedTimestampedRecord:

```
{
   "signal" : "vehicle.Body.ExteriorMirrors.Heating.Status"
   "ts" : "2020-01-10T02:59:43.491751"
   "value" : "false"
}
```

TimeSeries:

```
{
    "signal"  :  "vehicle.body.cabin.temperature"
    "count"   : "132"  # Might be redundant information, optional.
    "values" : {
        {
            "ts" : "2020-01-10T02:59:43.491751"
            "value" : "42.5"
        },
        {
            "ts" : "2020-01-10T02:59:43.491751"
            "value" : "43.0"
        },
        ... 130 more records
  }
}
```

Snapshot:

```
{
    "timeperiod" {
        "start" : "2020-01-10T02:00:00Z",
        "end" : "2020-01-11T01:59:59Z"
    },
    "values" : {
     {
        "signal" : "vehicle.body.cabin.temperature",
        "value" : "22.0",
        "ts" : "2020-01-10T02:59:43.491751"
     },
     {
        "signal" : "vehicle.drivetrain.engine.rpm.average",
        "value" : "3200",
        "ts" : "2020-01-10T02:59:44.100403"
     }
}
```

todo: Examples of Derived/Statistics Records

Other example representations

Above we used a simple JSON encoding for the data (as an example). A more space-efficient binary format is also possible, and here we can reuse existing technologies (AVRO, Protobuf, Thrift, CBOR, ...)

Note that the AVRO schemas are also written in JSON, so unlike the examples above this is **not** an *example of the data content* JSON is used here to describe how data *will* be structured.
The data content is stored/transferred by AVRO implementations according to the schema. It uses an efficient binary encoding for the values.

AVRO-schema example

```
{
  "type" : "record",
  "name" : "SpecifiedTimeStampedRecord",
  "fields" : [
    { "name" : "signal_identifier", "type" : "string" },
    { "name" : "ts", "type" : "long" },
    { "name" : "value", "type" : "Value" }
  ]
```

... where **Value** is a union of all the possible VSS types.
This is a little convoluted because values can be any plain data type, or an Array of such datatypes:

```
{
  "type" : "record",
  "name" : "Value",
  "fields" : [
    { "name" : "item", "type" : [
        "int", "long", "float", "double", "string", "boolean",
        { "type" : "array",
          "items" : [ "int", "long", "float", "double", "string", "boolean" ]
        }
      ]
    }
  ]
}
```

More AVRO encoding here: vss-tools (serializations branch)