VSS Working Sessions

COVESA AMM

2023-10-12

COVESA

Accelerating the future of connected vehicles

VSS Working Session

- Part 1
  - Walk through and discussion of recently presented VSS improvement areas
    - Based on presentations from Ford and Blackberry
- Part 2
  - VSS catalog evolvement
    - What to add, what to remove, …
  - VSS format and tooling
    - What changes do we see as wanted/needed?
    - What shall we NOT change

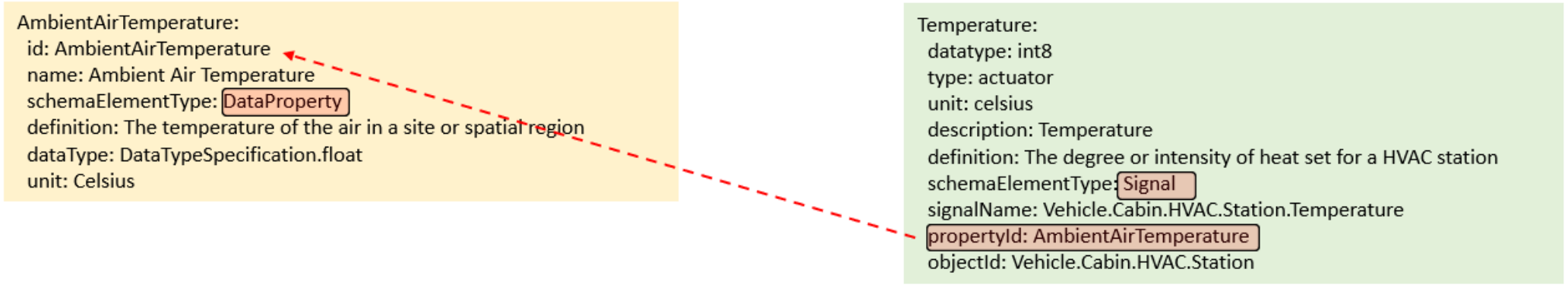
This is supposed to be an interactive session – please interrupt!

Typical workflow of a successful change to VSS

- Someone presents an idea
  - Creates an issue in VSS Github explaining the idea or problem area
  - Presents it at a VSS meeting
- We agree that the idea is good (or at least acceptable)
  - First discussed in Github and VSS Meetings
  - If needed also discussed/decided in DEG, TST, Board
- Someone volunteer to drive development
  - Creates more detailed proposals, possibly including prototypes
  - Propose time plan and acceptance criteria
  - Implements and creates Pull Requests
- Pull Requests reviewed and discussed by VSS meetings
  - When approved merged by VSS Maintainers (individuals with merge rights, typically Erik/Adnan/Sebastian)
  - Change included in next minor and/or major-release

VSS Type Representation

- VSS Today
  - Type of a signal is implicitly given by a combination of datatype/unit/min/max/allowed
  - “Type” reuse only possible for structs
- Idea presented
  - Define reusable properties

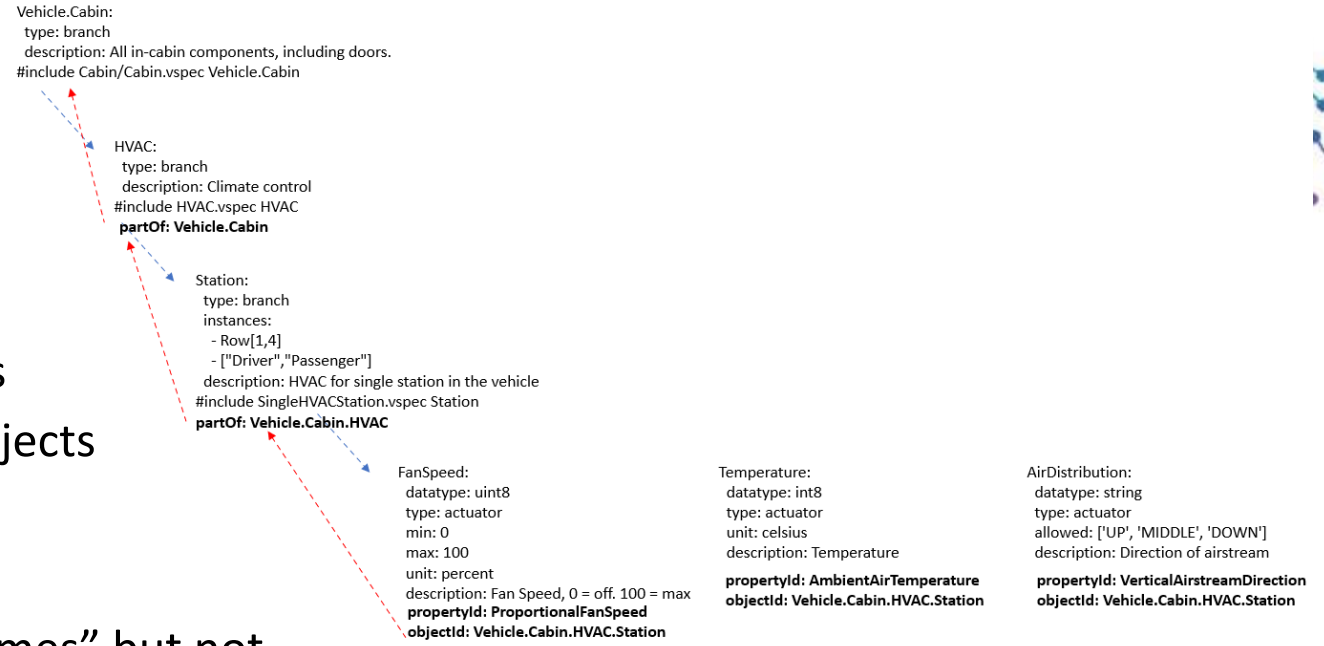


VSS Type Representation

- Pros/Cons
  - Makes VSS model somewhat more complex
  - Fits quite well into the type concept we already have for structs
  - Could help keeping VSS standard catalog consistent
    - Avoid that similar signals have different type/unit
    - *”All temperature signals shall use TemperatureCelsius property!*”
- Possible VSS Solutions
  - Extend existing type/struct syntax for this purpose
  - Convert existing signals to use properties
  - Tools must be modified – Are two flavors needed, one that keep property intformation and one that expand/replace property information to keep backward compatibility?
- Discussion – What is your opinion?

Tree vs Flat Structure

- VSS Today
  - One Root
  - Branches more or less used as namespaces
  - No real distinction between classes and objects
- Ford Idea (our interpretation)
  - No real requirement to have a tree
  - We may have “hierarchical dot notated names” but not necessarily
  - It is up to the child to declare it’s father rather than opposite
- Related comments
  - We have previously got comments that VSS expanded name are too long for some environments where identifiers have a max length.



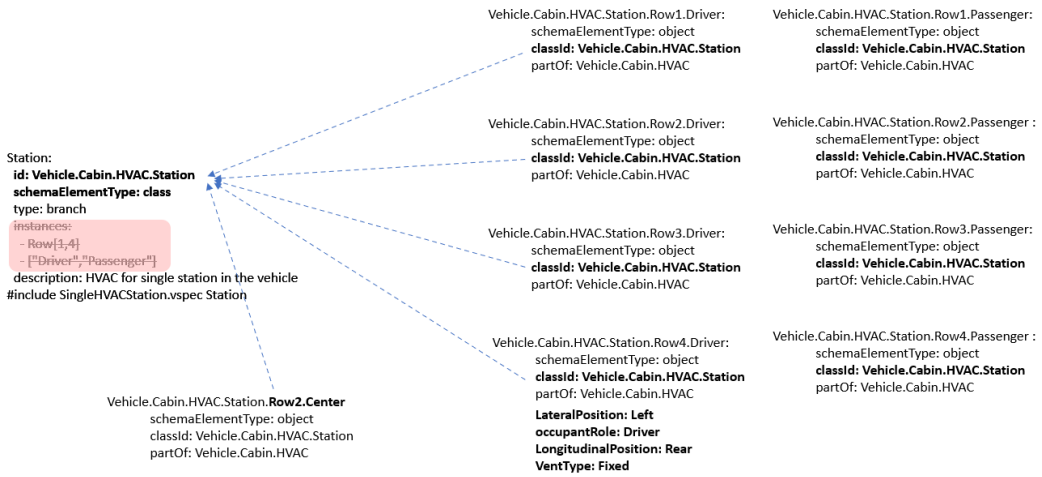
Tree vs Flat Structure continued

- Implications and Discussion Topics
  - This would be a big change if performed in *.vspec files
    - And if so, a good point in time to discuss if *.vspec shall be kept as source format
  - Generating “old style” expanded Yaml/JSON from new format doable, if needed
  - Even if we “scrap” branches – do we still some need some namespace mechanism and support for relative addressing?
- Discussion – What is your opinion?

Instances

- VSS Today
  - Instances defined on branch
  - Syntax limited, basically enum and/or array
- Ford Idea (our interpretation)
  - You define objects instead of specifying instances in the class
  - Arbitrary Identifier, not necessarily following VSS “expanded names”
  - Possibility to define static object data
    - Like “LateralPosition” in image
- Related comments
  - Some downstream projects like Eclipse Velocitas prefer t work with unexpanded paths to allow methods like “getHVAC(row,pos)”

MDP – Invert Tree Syntax - Instances



Instances continued

- Implications and Discussion Topics
  - Instances is common discussion point for VSS
  - The current style forces us to have “default size” for number of doors/seats/HVACs as you MUST specify instances at standard catalog level
  - Current VSS solution has some possibility to add signals for a specific instance by overlays specified by expanded name. Would require a different solution if using this syntax (class inheritance to allow addition of signals?)
  - Do we need a special mechanism to specify that “LateralPosition” is a “const signal/attribute” of HVACStation, i.e. something that must be given when instantiating, i.e. something that an SDK can use to find a matching HVAC station?
- Discussion – What is your opinion?

Signal type

- VSS today
  - Signals are specified as attribute, sensor or actuator
  - These types are often confusing, as there may not be a real sensor behind, the value may be calculated
  - What a client can do may anyway be limited by access rights
  - But distinction may serve a purpose in deployments
    - An actuator has both a current value and a wanted value, sensor/attribute has only current value

```
IsRecirculationActive:
  datatype: boolean
  type: actuator
  description: Is recirculation active.
  id: Vehicle.Cabin.HVAC.IsRecirculationActive
  name: Vehicle.Cabin.HVAC.IsRecirculationActive
  schemaElementType: Signal
  propertyId: IsActive
  objectId: Vehicle.Cabin.HVAC.Recirculation
  definition: Indicates whether the cabin air is be
  isDefinedBy: FordMotorCompany, COVESA
  definitionSource: VehicleSignalSpecification
```

Signal type - continued

- Implications and Discussion Topics
  - Some distinction may be useful
  - Example: Vehicle.Speed is never intended to be actuable (we think). Hood.IsOpen may be it in some vehicles
    - I.e. no value in that an API generate a “setTargetSpeed(float value)”
  - Ontology guys sometimes differentiate between Observable Properties and Actuable Properties
  - Does anyone make a distinction between attribute and sensor in their implementation?
- Discussion – What is your opinion?

Description/Definition

- Ford proposal
  - Replace “description” with “definition”
- VSS Today
  - No real definition exists on what should be part of “description”
- Implications and Discussion Topics
  - Is the purpose of the current “description” and the intended “definition that different?
  - I.e. is just “definition” a better name, or do we need both?
- Discussion – What is your opinion?

```
IsRecirculationActive:
  datatype: boolean
  type: actuator
  description: Is recirculation active.
  id: Vehicle.Cabin.HVAC.IsRecirculationActive
  name: Vehicle.Cabin.HVAC.IsRecirculationActive
  schemaElementType: Signal
  propertyId: IsActive
  objectId: Vehicle.Cabin.HVAC.Recirculation
  definition: Indicates whether the cabin air is being recycled
  isDefinedBy: FordMotorCompany, COVESA
  definitionSource: VehicleSignalSpecification
```

Permission Model

- VSS Today
  - Does not specify anything, not even a syntax
- Implications and Discussion Topics
  - Access rights can likely never be specified in VSS standard catalog
  - But we could specify “recommended methods” where we define syntax allowing VSS models to be annotated
  - We could extend tooling to check for consistency
- Discussion – What is your opinion?

Permissions File

```
"x-read-permission": [
    {
        "permission": "Vehicle.READ",
        "nodes": [
            "Vehicle"
        ]
    },
    {
        "permission": "Vehicle.PII.READ",
        "nodes": [
            "Vehicle.VehicleIdentification",
            "Vehicle.Driver.Identifier"
        ]
    }
]
```

Signal Catalog File

```
"Speed": {
    "datatype": "float",
    "description": "Vehicle speed.",
    "type": "sensor",
    "unit": "km/h",
    "uuid": "efe50798638d55fab18ab7d43cc490e9",
    "x-euuid": "86e92e0ee67d30dd",
    "x-read-permission": "Vehicle.READ"
},
```

UUID

- VSS Today
  - We have UUID based on path name, but method not well described
  - No method to use other UUID, a different UUID method or to reuse old UUIDs and map UUIDs
- Implications and Discussion Topics
  - What use cases do we intend to solve
    - Short identifiers to reduce message length?
    - Mapping signals with same content
    - Identify changes compared to standard VSS or compared to previous version
  - Do we think one method would suffice?
  - Or do we think it will be use-case dependent?
- Discussion – What is your opinion?

PERMISSION SUPPORT AND EUUID GENERATION

Permissions File

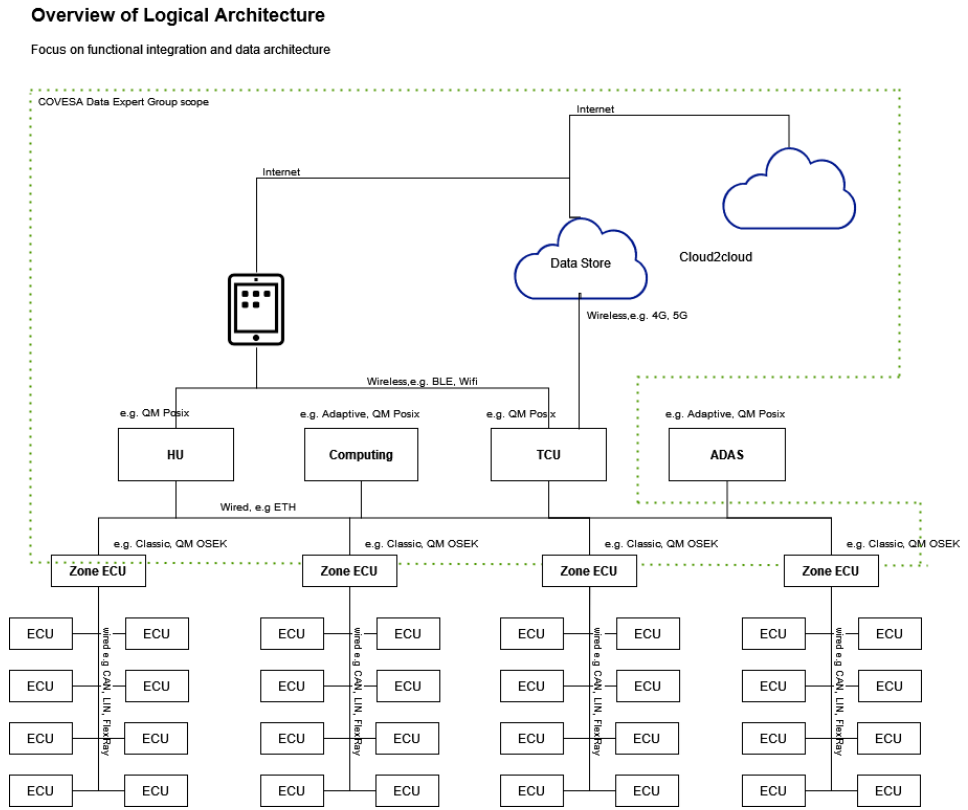
```
"x-read-permission": [
    {
        "permission": "Vehicle.READ",
        "nodes": [
            "Vehicle"
        ]
    },
    {
        "permission": "Vehicle.PII.READ",
        "nodes": [
            "Vehicle.VehicleIdentification",
            "Vehicle.Driver.Identifier"
        ]
    }
]
```

Signal Catalog File

```
"Speed": {
    "datatype": "float",
    "description": "Vehicle speed.",
    "type": "sensor",
    "unit": "km/h",
    "uuid": "efe50798638d55fab18ab7d43cc490e9",
    "x-euuid": "86e92e0ee67d30dd",
    "x-read-permission": "Vehicle.READ"
},
```

VSS Catalog evolvement – VSS History

- Focus on signals needed by “vehicle users”
  - infotainment, apps
  - Remote unlock, check fuel level, …
- QM only, i.e. not intended for safety critical features
  - Primarily not intended for “zones”, at least not with a generic key-value-based solution like KUKSA or VISS
- Shall be vehicle focused, but may be intended for both off-board/on-board usage.
  - Example: Vehicle color may be totally irrelevant to store on-board, but is relevant in a vehicle database
- Aggregated data on fleet level is not within the scope of VSS
  - Example: Average distance between accidents
- Even if VSS use the terms “actuator” and “sensor”, they do not necessarily map to a physical sensor/actuator
  - Actuator: Something where a user can set a target value
    - Example: Vehicle.HVAC.Temperature
  - Sensor: Something that typically varies over time, but a user cannot set target value directly
    - Example: Vehicle.Speed



VSS Catalog – Trends and Change Methodology

- Most low hanging fruit already picked, signals are becoming more complex
  - Just studying an individual signal is not enough, you need to study some concept documentation
  - Examples: PowerOptimize, WiperSystem
- Smaller non-controversial changes
  - Just make a Pull Request
- Larger changes
  - Create Issue/PR for discussion
  - If needed, create a COVESA task group or project with members interested in this topic:

VSS Catalog – Recent discussions - OBD

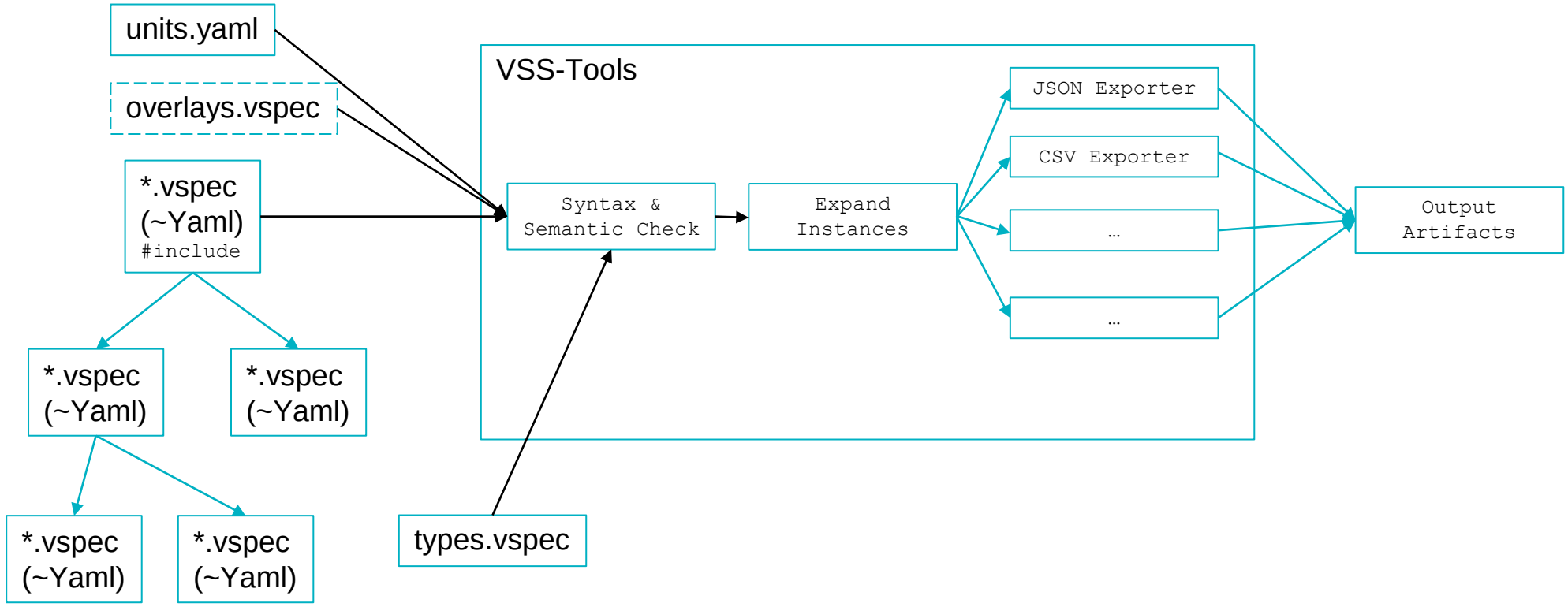
- VSS has an OBD branch which often triggers discussion
- Arguments Against
  - VSS and VSS-implementations (like VISS, KUKSA) are likely not feasible to provide data on the OBD-II interface due to latency/frequency requirements
  - Many signals in the branch are duplicates of similar signals in other parts of the tree
  - VSS shall not include 1:1 copies of other standards
- Arguments For
  - OBD-tools are expensive, making data like Diagnostic Codes (DTC) available over VSS could be useful for vehicle owners, so they can view data in an App on their mobile phone
- One possible solution (which seems to be preferred/acceptable by many)
  - Remove OBD-branch
  - But first add relevant signals to other places in VSS Tree
    - DTCs are frequently mentioned
    - But are others of interest, like lambda sensor readings?
- Discussion – What is your opinion?

VSS Catalog – Other improvements Areas

- We recently had a PR addressing the “Driver branch”
  - Fatigue, Attentiveness, Eyes on road, …
  - That branch appears to not be that mature, and if there are multiple parties interested this could be a candidate for a task force/sub-project to identify needed changes.
- Discussion – What other areas do you see where changes would be beneficial?
  - Any areas you would be interested in driving?

VSS Format and Tooling

- Current Python Tool-Chain
- Includes some variation points
  - With/Without UUID
  - Expansion of instances (or not)



VSS Format and Tooling

- The use of *.vspec (basically Yaml with some extensions) as source format is sometimes challenged
  - Does not fit that well with “ontology” representation, or the representation proposed by Ford
  - Limited out of the box support, like schema definition
- Some ideas presented would require major changes to VSS, i.e. if we think that we need some other source format now might be a good time to do it.
- We always strive to minimize amount of changes that are backward incompatible
  - But we do not really know what all downstream projects use as input
  - So difficult to say what the consequences would be
- Examples:
  - Eclipse KUKSA use JSON generated by vss-tools as input. As long as we can generate JSON that looks like today the actual VSS source format (today *.vspec) does not matter
- Two major approaches to support more complex models
  - Keep *.vspec as of today, annotate it if needed so that vss-tools can generate more complex model, for example generating classes/objects/properties
    - Drawback: Makes tooling more complex
  - Replace *.vspec with “something else”. Derive *.vspec and/or current export results (CSV, JSON, …) from it.
    - Drawback: Completely new tooling required
- What is important for you concerning VSS Format and Tools?

COVESA

Accelerating the future of connected vehicles