



Sensors and actuators

1 Contents

2	Terminology and concepts	4
3	Vehicle	4
4	Intra-vehicle network	4
5	Inter-vehicle network	4
6	Sensor	5
7	Actuator	5
8	Device	5
9	Use cases	5
10	Augmented reality parking	5
11	Virtual mechanic	5
12	Petrol station finder	6
13	Sightseeing application bundle	6
14	Changing bundle functionality when driving at speed	6
15	Changing audio volume with vehicle or cabin noise	6
16	Night mode	7
17	Weather feedback or traffic jam feedback	7
18	Insurance bundle	7
19	Driving setup bundle	7
20	Odour detection	8
21	Air conditioning control	8
22	Agricultural vehicle	8
23	Roof box	8
24	Truck installations	8
25	Compromised application bundle	9
26	Ethernet intra-vehicle network	9
27	Development against the SDK	9
28	Non-use-cases	9
29	Bluetooth wrist watch and the Internet of Things	9
30	Car-to-car and car-to-infrastructure communications	10
31	Buddied and vehicle fleet communications	10
32	Requirements	11
33	Enumeration of devices	11
34	Enumeration of vehicles	11
35	Retrieving data from sensors	11
36	Sending data to actuators	11
37	Network independence	11
38	Bounded latency of processing sensor data	12
39	Extensibility for OEMs	12
40	Third-party backends	12
41	Third-party backend validation	12
42	Notifications of changes to sensor data	12
43	Uncertainty bounds	13
44	Failure feedback	13
45	Timestamping	13

46	Triggering bundle activation	13
47	Bulk recording of sensor data	14
48	Sensor security	14
49	Actuator security	14
50	App store knowledge of device requirements	14
51	Accessing devices on multiple vehicles	14
52	Third-party accessories	15
53	SDK hardware support	15
54	Background on intra-vehicle networks	15
55	Existing sensor systems	15
56	W3C Vehicle Information Service Specification (VISS)	16
57	GENIVI Web API Vehicle	16
58	Apple HomeKit	16
59	Apple External Accessory API	17
60	iOS CarPlay	18
61	Android Auto	18
62	MirrorLink	18
63	Android Sensor API	19
64	Automotive Message Broker	19
65	AllJoyn	20
66	Approach	20
67	Overall architecture	21
68	Vehicle device daemon	21
69	Hardware and app APIs	22
70	Hardware API compliance testing	26
71	SDK API compliance testing and simulation	27
72	SDK hardware	27
73	Trip logging of sensor data	28
74	Properties vs devices	28
75	Property naming	29
76	High bandwidth or low latency sensors	29
77	Timestamps and uncertainty bounds	30
78	Registering triggers and actions	30
79	Bulk recording of sensor data	31
80	Security	31
81	Suggested roadmap	38
82	Requirements	38
83	Open questions	40
84	Summary of recommendations	41
85	Sensors and Actuators API	42
86	Rhosydd API Current State	42
87	Considerations to align Rhosydd to the new VISS API	42
88	New vs Old Specification	42
89	Rhosydd New Changes	43
90	Advantages	44

91	Conclusion	44
92	Appendix: W3C API	44

93 This documents possible approaches to designing an API for exposing vehicle
 94 sensor information and allowing interaction with actuators to application bun-
 95 dles on an Apertis system.

96 The major considerations with a sensors and actuators API are:

- 97 • Bandwidth and latency of sensor data such as that from parking cameras
- 98 • Enumeration of sensors and actuators
- 99 • Support for multiple vehicles or accessories
- 100 • Support for third-party and OEM accessories and customisations
- 101 • Multiplexing of access to sensors
- 102 • Privilege separation between application bundles using the API
- 103 • Policy to restrict access to sensors (privacy sensitive)
- 104 • Policy to restrict access to actuators (safety critical)

105 Terminology and concepts

106 Vehicle

107 For the purposes of this document, a *vehicle* may be a car, car trailer, motor-
 108 bike, bus, truck tractor, truck trailer, agricultural tractor, or agricultural trailer,
 109 amongst other things.

110 Intra-vehicle network

111 The *intra-vehicle network* connects the various devices and processors through-
 112 out a vehicle. This is typically a CAN or LIN network, or a hierarchy of such
 113 networks. It may, however, be based on Ethernet or other protocols.

114 The vehicle network is defined by the OEM, and is statically defined —all devices
 115 which are supported by the network have messages or bandwidth allocated for
 116 them at the time of manufacture. No devices which are not known at the time
 117 of manufacture can be supported by the vehicle network.

118 Inter-vehicle network

119 An *inter-vehicle network* connects two or more *physically connected* vehicles
 120 together for the purposes of exchanging information. For example, a network
 121 between a truck tractor and trailer.

122 An inter-vehicle network (for the purposes of this document) does *not* cover
 123 transient communications between separate cars on a motorway, for example;
 124 or between a vehicle and static roadside infrastructure it passes. These are

125 car-to-car (C2C) and car-to-infrastructure (C2X) communications, respectively,
126 and are handled separately.

127 **Sensor**

128 A *sensor* is any input device which is connected to the vehicle's network but
129 which is not a direct part of the dashboard user interface. For example: parking
130 cameras, ultrasonic distance sensors, air conditioning thermometers, light level
131 sensors, etc.

132 **Actuator**

133 An *actuator* is any output device which is connected to the vehicle's network
134 but which is not a direct part of the dashboard user interface. For example:
135 air conditioning heater, door locks, electric window motors, interior lights, seat
136 height motors, etc.

137 **Device**

138 A sensor or actuator.

139 **Use cases**

140 A variety of use cases for application bundle usage of sensor data are given
141 below. Particularly important discussion points are highlighted at the bottom
142 of each use case.

143 **Augmented reality parking**

144 When parking, the feed from a rear-view camera should be displayed on the
145 screen, with an overlay showing the distance between the back of the vehicle
146 and the nearest object, taken from ultrasonic or radar distance sensors.

147 The information from the sensors has to be synchronised with the camera, so
148 correct distance values are shown for each frame. The latency of the output
149 image has to be low enough to not be noticed by the driver when parking at
150 low speeds (for example, 5km·h).

151 **Virtual mechanic**

152 Provide vehicle status information such as tyre pressure, engine oil level, washer
153 fluid level and battery status in an application bundle which could, for example,
154 suggest routine maintenance tasks which need to be performed on the vehicle.

155 (Taken from http://www.w3.org/2014/automotive/vehicle_spec.html#h2_abst
156 *ract.*)

157 **Trailer** The driver attaches a trailer to their vehicle and it contains tyre pres-
158 sure sensors. These should be available to the virtual mechanic bundle.

159 **Petrol station finder**

160 Monitor the vehicle's fuel level. When it starts to get low, find nearby petrol
161 stations and notify the driver if they are near one. Note that this requires
162 programs to be notified of fuel level changes while not in the foreground.

163 **Sightseeing application bundle**

164 An application bundle could highlight sights of interest out of the windows by
165 combining the current location (from GPS) with a direction from a compass
166 sensor. Using a compass rather than the GPS'velocity angle allows the bundle
167 to work even when the vehicle is stationary.

168 **Privacy concern:** Any application bundle which has access to compass data
169 can potentially use dead reckoning to track the vehicle's location, even without
170 access to GPS data.

171 **Basic model vehicle** If a vehicle does not have a compass sensor, the sight-
172 seeing bundle cannot function at all, and the Apertis store should not allow the
173 user to install it on their vehicle.

174 **Changing bundle functionality when driving at speed**

175 An application bundle may want to voluntarily change or disable some of its
176 features when the vehicle is being driven (as opposed to parked), or when it
177 is being driven fast (above a cut-off speed). It might want to do this to avoid
178 distracting the driver, or because the features do not make sense when the
179 vehicle is moving. This requires bundles to be able to access speedometer and
180 driving mode information.

181 If the application bundle is using a cut-off speed for this decision, it should not
182 have to continually monitor the vehicle's speed to determine whether the cut-off
183 has been reached.

184 **Changing audio volume with vehicle or cabin noise**

185 Bundles may want to adjust their audio output volume, or disable audio output
186 entirely, in response to changes in the vehicle's cabin or engine noise levels. For
187 example, a game bundle could reduce its effects volume if a loud conversation
188 can be heard in the cabin; but it might want to increase its effects volume if
189 engine noise increases.

190 **Privacy concern:** This should be implemented by granting access to overall
191 'volume level' information for different zones in the vehicle; but *not* by grant-
192 ing access to the actual audio input data, which would allow the bundle to

193 record conversations. The overall volume level information should be sufficiently
194 smoothed or high-latency that a malicious application cannot infer audio infor-
195 mation from it.

196 **Night mode**

197 Programs may wish to change their colour scheme according to the ambient
198 lighting level in a particular zone in the cabin, for example by switching to a
199 ‘night mode’ with a dark colour scheme if driving at night, but not if an interior
200 light is on. This requires bundles to be able to read external light sensors and
201 the state of internal lights.

202 **Weather feedback or traffic jam feedback**

203 A weather bundle may want to crowd-source information about local weather
204 conditions to corroborate its weather reports. Information from external rain,
205 temperature and atmospheric pressure sensors could be collected at regular in-
206 tervals –even while the weather bundle is not active –and submitted to an online
207 weather service as network connectivity permits.

208 Similarly, a traffic jam or navigation bundle may want to crowd-source informa-
209 tion about traffic jams, taking input from the speedometer and vehicle separa-
210 tion distance sensors to report to an online service about the average speed and
211 vehicle separation in a traffic jam.

212 **Insurance bundle**

213 A vehicle insurance company may want to offer lower insurance premiums to
214 drivers who install its bundle, if the bundle can record information about their
215 driving safety and submit it to the insurance company to give them more infor-
216 mation about the driver’s riskiness. This would need information such as driving
217 duration, distances driven, weather conditions, acceleration, braking frequency,
218 frequency of using indicator lights, pitch, yaw and roll when cornering, and
219 potentially vehicle maintenance information. It would also require access to
220 unique identifiers for the vehicle, such as its VIN. The data would need to be
221 collected regardless of whether the vehicle is connected to the internet at the
222 time –so it may need to be stored for upload later.

223 **Privacy concern:** Unique identification information like a VIN should not be
224 given to untrusted bundles, as they may use it to track the user or vehicle.

225 **Driving setup bundle**

226 An application bundle may want to control the driving setup –the position of the
227 steering wheel, its rake, the position of the wing mirrors, the seat position and
228 shape, whether the vehicle is in sport mode, etc. If a guest driver starts using
229 the vehicle, they could import their settings from the same bundle on their own
230 vehicle, and the bundle would automatically adjust the physical driving setup

231 in the vehicle to match the user's preferences. The bundle may want to restrict
232 these changes to only happen while the vehicle is parked.

233 **Odour detection**

234 A vehicle manufacturer may have invented a new type of interior sensor which
235 can detect foul odours in the cabin. They want to integrate this into an ap-
236 plication bundle which will change the air conditioning settings temporarily to
237 clear the odour when detected. The Sensors and Actuators API currently has
238 no support for this new sensor. The manufacturer does not expect their bundle
239 to be used in other vehicles.

240 **Air conditioning control**

241 An application bundle which connects to wrist watch body monitors on each
242 of the passengers (through an out-of-band channel like Bluetooth, which is out
243 of the scope of this document; see [Bluetooth wrist watch and the Internet of
244 Things](#) may want to change the cabin temperature in response to thermometer
245 readings from passengers' watches.

246 **Automatic window feedback** In order to do this, the bundle may also need
247 to close the automatic windows, but one of the passengers has their arm hanging
248 out of the window and the hardware interlock prevents it closing. The bundle
249 must handle being unable to close the window.

250 **Agricultural vehicle**

251 Apertis is used by an agricultural manufacturer to provide an IVI system for
252 drivers to use in their latest tractor model. The manufacturer provides a pre-
253 installed app for controlling their own brand of agricultural accessories for the
254 tractor, so the driver can use it to (for example) control a tipping trailer and
255 a baler which are hitched to each other behind the tractor, and also control a
256 bale spear attached to the front of the tractor.

257 **Roof box**

258 A car driver adds a roof box to their car, provided by a third party, containing
259 a safety sensor which detects when the box is open. The built-in application
260 bundle for alerting the driver to doors which are open when the vehicle starts
261 moving should be able to detect and use this sensor to additionally alert the
262 driver if the roof box is open when they start moving.

263 **Truck installations**

264 Trucks are sold as a basis 'vanilla' truck with a special installation on top, which
265 is customised for the truck's intended use. For example, a rubbish truck, tipping
266 truck or police truck. The installation is provided by a third party who has a

267 relationship with the basis truck manufacturer. Each installation has specific
268 sensors and actuators, which are to be controlled by an application bundle
269 provided by the third party or by the manufacturer.

270 **Compromised application bundle**

271 An application bundle on the system, A, is installed with permissions to adjust
272 the driver's seat position, which is one of the features of the bundle. Another
273 application bundle, B, is installed without such permissions (as they are not
274 needed for its normal functionality).

275 **Safety critical:** An attacker manages to exploit bundle B and execute arbitrary
276 code with its privileges. The attacker must not be able to escalate this exploit
277 to give B permission to use actuators attached to the system, or extra sensors.
278 Similarly, they must not be able to escalate the exploit to gain the privileges of
279 bundle A, and hence bundle A's permissions to adjust the driver's seat position.

280 **Ethernet intra-vehicle network**

281 A vehicle manufacturer wants to support high-bandwidth devices on their intra-
282 vehicle network, and decides to use Ethernet for all intra-vehicle communica-
283 tions, instead of a more traditional CAN or LIN network. Their use of a differ-
284 ent network technology should not affect enumeration or functionality of devices
285 as seen by the user.

286 **Development against the SDK**

287 An application developer wants to use a local gyroscope sensor attached to their
288 development machine to feed input to their application while they are developing
289 and testing it using the SDK.

290 **Non-use-cases**

291 **Bluetooth wrist watch and the Internet of Things**

292 A passenger gets into the vehicle with a Bluetooth wrist watch which monitors
293 their heart rate and various other biological variables. They launch their health
294 monitor bundle on the IVI display, and it connects to their watch to download
295 their recent activity data.

296 This is not a use case for the Sensors and Actuators API; it should be handled
297 by direct Bluetooth communication between the health monitor bundle and the
298 watch. If the Sensors and Actuators API were to support third-party devices
299 (as opposed to ones specified and installed by the vehicle manufacturer or sup-
300 pliers), having full support for all available devices would become a lot harder.
301 Additionally, devices would then appear and disappear while the vehicle was
302 running (for example, if the user turned off their watch's Bluetooth connection

303 while driving); this is not possible with fixed in- vehicle sensors, and would
304 complicate the sensor enumeration API.

305 More generally, this use-case is a specific case of the internet of things (IoT),
306 which is out of scope for this design for the reasons given above. Additionally,
307 supporting IoT devices would mean supporting wireless communications as part
308 of the sensors service, which would significantly increase its attack surface due
309 to the complexity of wireless communications, and the fact they enable remote
310 attacks.

311 **Car-to-car and car-to-infrastructure communications**

312 In C2C and C2X communications, vehicles share data with each other as they
313 move into range of each other or static roadside infrastructure. This information
314 may be anything from braking and acceleration information shared between
315 convoys of vehicles to improve fuel efficiency, to payment details shared from a
316 car to toll booth infrastructure.

317 While many of the use cases of C2C and C2X cover sharing of sensor data, the
318 data being shared is typically a limited subset of what's available on one vehicle'
319 s network. Due to the dynamic nature of C2C and C2X networks, and the
320 greater attack surface caused by the use of more complex technologies (radio
321 communications rather than wired buses), a conservative approach to security
322 suggests implementing C2C and C2X on a use-case-by-use-case basis, using sep-
323 arate system components to those handling intra-vehicle sensors and actuators.
324 This ensures that control over actuators, which is safety critical, remains in a
325 separate security domain from C2C and C2X, which must not have access to
326 actuators on the local vehicle. See [Security](#).

327 An initial suggestion for C2C and C2X communications would be to implement
328 them as a separate service which consumes sensor data from the sensors and
329 actuators service just like other applications.

330 **Buddied and vehicle fleet communications**

331 Similarly, long-range communications of sensor data between buddied vehicles
332 or vehicles operating in a fleet (for example, a haulage or taxi fleet) should
333 be handled separately from the sensors and actuators service, as such systems
334 involve network communications. Typical use cases here would be reporting
335 speed and fuel usage information from trucks or taxis back to headquarters; or
336 letting two friends know each others'locations and traffic conditions when both
337 doing the same journey.

338 **Requirements**

339 **Enumeration of devices**

340 An application bundle must be able to enumerate devices in the vehicle, includ-
341 ing information about where they are located in the vehicle (for example, so
342 that it can adjust the position and setup of the driver's seat but not others (see
343 [Driving setup bundle](#))).

344 It is expected that the set of devices in a vehicle may change dynamically while
345 the vehicle is running, for example if a roof box were added while the engine
346 was running ([Roof box](#)).

347 Enumeration is particularly important for bundles, as the set of sensors in a
348 particular vehicle will not change, but the set of sensors seen by a bundle across
349 all the vehicles it's installed in will vary significantly.

350 **Enumeration of vehicles**

351 An application bundle must be able to enumerate vehicles connected to the
352 inter-vehicle network, for example to discover the existence of hitched trailers
353 or agricultural vehicles ([Trailer](#), [Agricultural vehicle](#)).

354 It is expected that the set of vehicles may change dynamically while the vehicles
355 are running.

356 **Retrieving data from sensors**

357 An application bundle must be able to retrieve data from sensors. This data
358 must be strongly typed in order to minimise the possibility of a bundle mis-
359 interpreting it, or sensors from different manufacturers using different units,
360 for example. Sensor data could vary in type from booleans (see [Night mode](#))
361 through to streaming video data (see [Augmented reality parking](#)). Sensor data
362 may be processed by the system to make it more useful for application bundles;
363 they do not need direct access to raw sensor data.

364 **Sending data to actuators**

365 An application bundle must be able to send data to actuators (including invok-
366 ing methods on them). This data must be strongly typed in order to minimise
367 the possibility of a bundle misinterpreting it, or actuators from different man-
368 ufacturers using different units, for example. Actuator data could vary in type
369 from booleans through to enumerated types (see [Driving setup bundle](#)) and
370 possibly larger data streams, though no concrete use cases exist for that.

371 **Network independence**

372 The API should be independent of the network used to connect to devices –
373 whether it be Ethernet, LIN or CAN; or whether the device is connected directly

374 to a host processor ([Ethernet intra-vehicle network](#)).

375 **Bounded latency of processing sensor data**

376 Certain sensor data has bounds on its latency. For example, pitch, yaw and
377 roll information typically arrive as angular rate from sensors, and have to be
378 integrated over time to be useful to application bundles –if sensor readings are
379 missed, accuracy decreases. Sensor readings should be processed within the
380 latency limits specified by the sensors. The limits on forwarding this processed
381 data to bundles are less strict, though it is expected to be within the latency
382 noticeable by humans (around 20ms) so that it can be displayed in real time
383 (see [Augmented reality parking](#), [Sightseeing application bundle](#), [Changing audio
384 volume with vehicle or cabin noise](#)).

385 **Extensibility for OEMs**

386 New types of device may be developed after the Sensors and Actuators API is
387 released. As the set of sensors in a vehicle does not vary after release, already-
388 deployed versions of the API do not need to handle unknown devices. However,
389 there must be a mechanism for OEMs or third parties working with them to
390 define new device types when developing a new vehicle or an installation or
391 accessory to go with it. In order for new devices to be usable by non-OEM
392 application bundle authors, the Sensors and Actuators API must be updatable
393 or extensible to support them. ([Odour detection](#), [Truck installations](#).)

394 **Third-party backends**

395 If an OEM or third party produces a new device which can be connected to
396 an existing vehicle, some code needs to exist to allow communication between
397 the device and the Apertis sensors and actuators service. This code must be
398 written by the device manufacturer, as they know the hardware, and must be
399 installable on the Apertis system before or after vehicle production (so as a
400 system or non-system application). (See [Agricultural vehicle](#), [Roof box](#), [Truck
401 installations](#).)

402 **Third-party backend validation**

403 If a third-party device is exposed to the sensors and actuators service, the third
404 party might not be one who has contributed to or used Apertis before. There
405 must be a process for validating backends for the sensors and actuators system,
406 to ensure they have a certain level of code quality and security, in order to
407 reduce the attack surface of the service as a whole. (See [Roof box](#).)

408 **Notifications of changes to sensor data**

409 All sensor data changes over time, so the API must support notifying application
410 bundles of changes to sensor data they are interested in, without requiring the

411 bundle to poll for updates (see [Petrol station finder](#), [Sightseeing application](#)
412 [bundle](#), [Changing bundle functionality when driving at speed](#), [Changing audio](#)
413 [volume with vehicle or cabin noise](#), [Night mode](#), [Odour detection](#)).

414 Application bundles should be able to request notifications only when a sensor
415 value crosses a given threshold, to avoid unnecessary notifications (see [Changing](#)
416 [bundle functionality when driving at speed](#)).

417 **Uncertainty bounds**

418 Sensors are not perfectly accurate, and additionally a sensor's accuracy may
419 vary over time; each sensor measurement should be provided with uncertainty
420 bounds. For example, the accuracy of geolocation by mobile phone tower varies
421 with your location.

422 This is especially possible with data aggregated from multiple sensors, where
423 the aggregate accuracy can be statistically modelled (for example, distance cal-
424 culation from multiple sensors in [Weather feedback or traffic jam feedback](#)).

425 **Failure feedback**

426 As actuators are physical devices, they can fail. The API cannot assume au-
427 tomatic, immediate or successful application of its changes to properties, and
428 needs to allow for feedback on all property changes.

429 For example, the air conditioning coolant on an older vehicle might have leaked,
430 leaving the air conditioning system unable to cool the cabin effectively. Appli-
431 cation bundles which wish to set the temperature need to have feedback from a
432 thermometer to work out whether the temperature has reached the target value
433 (see [Air conditioning control](#)).

434 Another example is failure to close windows: [Automatic window feedback](#).

435 **Timestamping**

436 In-vehicle networks (especially Ethernet) may have variable latency. In order
437 to correlate measurements from multiple sensors on the end of connections of
438 varying latency, each measurement should have an associated timestamp, added
439 at the time the measurement was recorded (see [Augmented reality parking](#),
440 [Sightseeing application bundle](#)).

441 **Triggering bundle activation**

442 Various use cases require a bundle to be able to trigger actions based on sensor
443 data reaching a certain value, even if the program is not running at that time
444 (see [Petrol station finder](#), [Changing audio volume with vehicle or cabin noise](#),
445 [Odour detection](#)). This requires some operating system service to monitor a
446 list of trigger conditions even while the programs which set those triggers are

447 not running, and start the appropriate program so that it can respond to that
448 trigger.

449 **Bulk recording of sensor data**

450 Some bundles require to be able to regularly record sensor measurements, with
451 the intention of processing them (for example, uploading them to an online
452 service) at a later time (see [Weather feedback or traffic jam feedback](#), [Insurance
453 bundle](#)). This is not latency sensitive. As an optimisation, a system service
454 could record the sensor readings for them, to avoid waking up the programs
455 regularly.

456 Data recorded in this way must only be accessible to the application bundle
457 which requested it be recorded.

458 The requesting application bundle is responsible for processing the data period-
459 ically, and deleting it once processed. The system must be able to periodically
460 overwrite recorded data if running low on space.

461 **Sensor security**

462 As highlighted by the privacy concerns in several of the use cases ([Sightseeing
463 application bundle](#), [Changing audio volume with vehicle or cabin noise](#), [Insur-
464 ance bundle](#)), there are security concerns with allowing bundles access to sensor
465 data. The system must be able to restrict access to some or all types of sensor
466 data unless the user has explicitly granted a bundle access to it. Bundles with
467 access to sensor data must be in separate security domains to prevent privilege
468 escalation ([Compromised application bundle](#)).

469 **Actuator security**

470 Control of actuators is safety critical but not privacy sensitive (unlike sensors).
471 The system must be able to restrict write access to some or all types of actuator
472 unless the user has explicitly granted a bundle access to it. Bundles with access
473 to actuators must be in separate security domains to prevent privilege escalation
474 ([Compromised application bundle](#)).

475 **App store knowledge of device requirements**

476 The Apertis store must know which devices (sensors *and* actuators) an appli-
477 cation bundle requires to function, and should not allow the user to install a
478 bundle which requires a device their vehicle does not have, or the bundle would
479 be useless ([Basic model vehicle](#)).

480 **Accessing devices on multiple vehicles**

481 The API must support accessing properties for multiple vehicles, such as hitched
482 agricultural trailers ([Agricultural vehicle](#)) or car trailers ([Trailer](#)). These vehi-

483 cles may appear dynamically while the IVI system is running; for example, in
484 the case where the driver hitches a trailer with the engine running.

485 **Note:** This requirement explicitly does not support C2C or C2X, which are out
486 of scope of this document. (See [Car-to-car and car-to-infrastructure communi-](#)
487 [cations](#)).

488 **Third-party accessories**

489 The API must support accessing properties of third-party accessories —either
490 dynamically attached to the vehicle ([Roof box](#)) or installed during manufacture
491 ([Truck installations](#)).

492 **SDK hardware support**

493 The SDK must contain a backend for the system which allows appropriate
494 hardware which is attached to the developer’s machine to be used as sensors or
495 actuators for development and testing of applications (see [Development against](#)
496 [the SDK](#)).

497 This backend must not be available in target images.

498 **Background on intra-vehicle networks**

499 For the purposes of informing the interface design between the Sensors and
500 Actuators API and the underlying intra-vehicle network, some background in-
501 formation is needed on typical characteristics of intra-vehicle networks.

502 CAN and LIN are common protocols in use, though future development may
503 favour Ethernet or other protocols. In all cases, the OEM statically defines all
504 protocols, data structures, and devices which can be on the network. Bandwidth
505 is allocated for all devices at the time of manufacture; even for devices which
506 are only optionally connected to the network, either because they’re a premium
507 vehicle feature, or because they are detachable, such as trailers. In these cases,
508 data structures on the network relating to those devices are empty when the
509 devices are not connected.

510 Sometimes flags are used in the protocol, such as ‘is a trailer connected?’.

511 There are no common libraries for accessing vehicle networks: they differ be-
512 tween OEMs.

513 **Existing sensor systems**

514 This chapter describes the approaches taken by various existing systems for
515 exposing sensor information to application bundles, because it might be useful
516 input for Apertis’s decision making. Where available, it also provides some details
517 of the implementations of features that seem particularly interesting or relevant.

518 **W3C Vehicle Information Service Specification (VISS)**

519 The [W3C Vehicle Information Service Specification](#)¹ defines a WebSocket based
520 API for a Vehicle Information Service (VIS) to enable client applications to
521 get, set, subscribe and unsubscribe to vehicle signals and data attributes. This
522 specification defines a number of methods for accessing vehicle data which are
523 strictly agnostic to the data model [Vehicle Signal Specification](#)².

524 The Vehicle Signal Specification (VSS) focuses on vehicle signals, in the sense
525 of classical sensors and actuators with the raw data communicated over vehi-
526 cle buses and data which is more commonly associated with the infotainment
527 system alike. This defines a 'tree-like' logical taxonomy of the vehicle, (formally
528 a Directed Acyclic Graph), where major vehicle structures (e.g. body, engine)
529 are near the top of the tree and the logical assemblies and components that
530 comprise them, are defined as their child nodes.

531 The VSS supports both extensibility and the ability to define private branches.

532 **GENIVI Web API Vehicle**

533 The [GENIVI Web API Vehicle](#)³ (sic) is a proof of concept API for exposing and
534 manipulating vehicle information to GENIVI apps via a JavaScript API. It is
535 very similar to the W3C Vehicle Information Access API, and seems to expose
536 a very similar set of properties.

537 The [Web API Vehicle](#)⁴ is a proxy for exposing a separate Vehicle Interface API
538 within a HTML5 engine. The Vehicle Interface API itself is apparently a D-Bus
539 API for sharing vehicle information between the CAN bus and various clients,
540 including this Web API Vehicle and any native apps. Unfortunately, the Vehicle
541 Interface API seems to be unspecified as of August 2015, at least in publicly
542 released GENIVI documents.

543 The Web API Vehicle has the same features and scope as the W3C API, but its
544 implementation is clumsier, relying a lot more on seemingly unstructured magic
545 strings for accessing vehicle properties.

546 It was last publicly modified in May 2013, and might not be under development
547 any more. Furthermore, a lot of the wiki links in the specification link to private
548 and inaccessible data on collab.genivi.org.

549 **Apple HomeKit**

550 [Apple HomeKit](#)⁵ is an API to allow apps on Apple devices to interact with
551 sensors and actuators in a home environment, such as garage doors, thermostats,

¹<https://www.w3.org/TR/vehicle-information-service/>

²https://github.com/COVESA/vehicle_signal_specification

³<https://at.projects.genivi.org/wiki/display/PROJ/Web+API+Vehicle>

⁴<https://at.projects.genivi.org/wiki/display/PROJ/Web+API+Vehicle>

⁵<https://developer.apple.com/homekit/>

552 thermometers and light switches, amongst others. It is designed explicitly for the
553 home environment, and does not consider vehicles. However, as it is effectively
554 an API for allowing interactions between sandboxed apps and external sensors
555 and actuators, it bears relevance to the design of such an API for vehicles.

556 At its core, HomeKit allows enumeration of devices ('accessories') in a home.
557 A large part of its API is dedicated to grouping these into homes, rooms, ser-
558 vice groups and zones so that collections of accessories can be interacted with
559 simultaneously.

560 Each accessory implements one or more 'services' which are defined interfaces
561 for specific functionality, such as a light switch interface, or a thermostat inter-
562 face. Each service can expose one or more 'characteristics' which are readable
563 or writable properties of that interface, such as whether a light is on, the cur-
564 rent temperature measured by a thermostat, or the target temperature for the
565 thermostat.

566 It explicitly maintains separation between *current* and *target* states for certain
567 characteristics, such as temperature controlled by a thermostat, acknowledging
568 that changes to physical systems take time.

569 A second part of the API implements 'actions' based on sensor values, which are
570 arbitrary pieces of code executed when a certain condition is met. Typically,
571 this would be to set the value of a characteristic on some actuator when the
572 input from another sensor meets a given condition. For example, switching on a
573 group of lights when the garage door state changes to 'open' as someone arrives
574 in the garage.

575 Critically, triggers and actions are handled by the iOS operating system, so are
576 still checked and executed when the app which created them is not active.

577 HomeKit has a [simulator](#)⁶ for developing apps against.

578 **Apple External Accessory API**

579 As a precursor to HomeKit, Apple also supports an [External Accessory API](#)⁷,
580 which allows any iOS device to interact with accessories attached to the device
581 (for example, through Bluetooth).

582 In order to use the External Accessory API, an app must list the accessory
583 protocols it supports in its app manifest. Each accessory supports one or more
584 protocols, defined by the manufacturer, which are interfaces for aspects of the
585 device's functionality. They are equivalent to the 'services' in the HomeKit API.
586 The code to implement these protocols is provided by the manufacturer, and
587 the protocols may be proprietary or standard.

⁶https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/HomeKitDeveloperGuide/TestingYourHomeKitApp/TestingYourHomeKitApp.html#//apple_ref/doc/uid/TP40015050-CH7-SW1

⁷<https://developer.apple.com/library/ios/featuredarticles/ExternalAccessoryPT/Introduction/Introduction.html>

588 Each accessory exposes [versioning information](#)⁸ which can be used to determine
589 the protocol to use.

590 All communication with accessories is done via [sessions](#)⁹, rather than one-shot
591 reads or writes of properties. Each session is a bi-directional stream along which
592 the accessory's protocol is transmitted.

593 **iOS CarPlay**

594 iOS [CarPlay](#)¹⁰ is a system for connecting an iOS device to a car's IVI system,
595 displaying apps from the phone on the car's display and allowing those apps to
596 be controlled by the car's touchscreen or physical controls. It *does not give*¹¹
597 the iOS device access to car sensor data, and hence is not especially relevant to
598 this design.

599 It *does not*¹² (as of August 2015) have an API for integrating apps with the IVI
600 display.

601 Most vehicle manufacturers have pledged support for it in the coming years.

602 **Android Auto**

603 [Android Auto](#)¹³ is very similar to iOS CarPlay: a system for connecting a phone
604 to the vehicle's IVI system so it can use the display and touchscreen or physical
605 controls. As with CarPlay, it *does not* give the Android device access to vehicle
606 sensor data, although (as of August 2015) that is planned for the future.

607 As of August 2015, it *has an API for apps*¹⁴, allowing audio and messaging apps
608 to improve their integration with the IVI display.

609 Most vehicle manufacturers have pledged support for it in the coming years.

610 **MirrorLink**

611 [MirrorLink](#)¹⁵ is a proprietary system for integrating phones with the IVI display
612 –it is similar to iOS CarPlay and Android Auto, but produced by the [Car](#)
613 [Connectivity Consortium](#)¹⁶ rather than a device manufacturer like Apple or
614 Google.

⁸https://developer.apple.com/library/ios/documentation/ExternalAccessory/Reference/EAAccessory_class/index.html#//apple_ref/occ/instp/EAAccessory/modelNumber

⁹https://developer.apple.com/library/ios/documentation/ExternalAccessory/Reference/EASession_class/index.html#//apple_ref/occ/instp/EASession/accessory

¹⁰<http://www.apple.com/uk/ios/carplay/>

¹¹<http://www.tomsguide.com/us/apple-carplay-faq,news-18450.html>

¹²<https://developer.apple.com/carplay/>

¹³<https://www.android.com/auto/>

¹⁴<https://developer.android.com/training/auto/index.html>

¹⁵<http://www.mirrorlink.com/apps>

¹⁶<http://carconnectivity.org/>

615 The specifications for MirrorLink are proprietary and only available to registered
616 developers. In a brochure (now unavailable for download), it is stated that
617 support for allowing apps access to sensor data is planned for the future (as of
618 2014).

619 MirrorLink is apparently the technology behind Microsoft's [Windows in the](#)
620 [Car](#)¹⁷ system, which was announced in April 2014.

621 **Android Sensor API**

622 [Android's Sensor API](#)¹⁸ is a mature system for accessing mobile phone sensors.
623 There are a more constrained set of sensors available in phones than in vehi-
624 cles, hence the API exposes individual sensors, each implementing an interface
625 specific to its type of sensor (for example, accelerometer, orientation sensor or
626 pressure sensor). The API places a lot of emphasis on the physical limitations of
627 each sensor, such as its range, resolution, and uncertainty of its measurements.

628 The sensors required by an app are listed in its manifest file, which allows the
629 Google Play store to filter apps by whether the user's phone has all the necessary
630 sensors.

631 As Android runs on a multitude of devices from different manufacturers, each
632 with different sensors, enumeration of the available sensors is also an emphasis
633 of the API, using its [SensorManager](#)¹⁹ class.

634 [Sensors](#)²⁰ can be queried by apps, or apps can register for notifications when
635 sensor values change, including when the app is not in the foreground or when
636 the device is asleep (if supported by the sensor). Apps can also [register](#)²¹ for no-
637 tifications when sensor values satisfy some trigger, such as a 'significant' change.

638 **Automotive Message Broker**

639 [Automotive Message Broker](#)²² is an Intel OTC project to broker information
640 from the vehicle networks to applications, exposing a [tweaked version](#)²³ of the
641 W3C Vehicle Information Access API (with a few types and naming conventions
642 tweaked) over D-Bus to apps, and interfacing with whatever underlying networks
643 are in use in the vehicle. In short, it has the same goals as the Apertis Sensors
644 and Actuators API.

¹⁷<http://www.techradar.com/news/car-tech/microsoft-sets-its-sights-on-apple-carplay-with-windows-in-the-car-concept-1240245>

¹⁸<http://developer.android.com/guide/topics/sensors/index.html>

¹⁹<http://developer.android.com/reference/android/hardware/SensorManager.html>

²⁰<http://developer.android.com/reference/android/hardware/SensorManager.html#registerListener%28android.hardware.SensorEventListener,%20android.hardware.Sensor,%20int%29>

²¹<http://developer.android.com/reference/android/hardware/SensorManager.html#requestTriggerSensor%28android.hardware.TriggerEventListener,%20android.hardware.Sensor%29>

²²<https://github.com/otcshare/automotive-message-broker>

²³<https://github.com/otcshare/automotive-message-broker/blob/master/docs/amb.in.fidl>

645 As of August 2015, it was last modified in June 2015, so is an active project
646 (although Tizen is in decline, so this may change). Although it is written in
647 C++, it uses GNOME technologies like GObject Introspection; but it also uses
648 Qt. Its main daemon is the Automotive Message Broker daemon, ambd.

649 One area where it differs from the Apertis design is **Security**; it does not im-
650 plement the polkit integration which is key to the vehicle device daemon secu-
651 rity domain boundary. Modifying the security architecture of a large software
652 project after its initial implementation is typically hard to get right.

653 Another area where ambd differs from the Apertis design is in the backend:
654 ambd uses multiple plugins to aggregate vehicle properties from many places.
655 Apertis plans to use a single OEM-provided, vehicle-specific plugin.

656 AllJoyn

657 The **AllJoyn Framework**²⁴ is an internet of things (IoT) framework produced
658 under the Linux Foundation banner and the **Open Connectivity Foundation**²⁵.
659 (Note that IoT frameworks are explicitly out of scope for this design; this section
660 is for background information only. See **Bluetooth wrist watch and the Internet
661 of Things**) It allows devices to discover and communicate with each other. It is
662 freely available (open source) and has components which run on various different
663 operating systems.

664 As a framework, it abstracts the differences between physical transports, provid-
665 ing a session API for devices to use in one-to-one or one-to-many configurations
666 for communication. A lot of its code is orientated towards implementing differ-
667 ent physical transports.

668 It provides a security API for establishing different trust models between devices.

669 It provides various communication layer APIs for implementing RPC or raw
670 I/O streams (or other things in-between) between devices. However, it does not
671 specify the protocols which devices must use —they are specified by the device
672 manufacturer.

673 AllJoyn provides common services for setting up new devices, sending notifica-
674 tions between devices, and controlling devices. It provides one example service
675 for controlling lamps in a house, where each lamp manufacturer implements
676 a well-defined OEM API for their lamp, and each application uses the lamp
677 service API which abstracts over these.

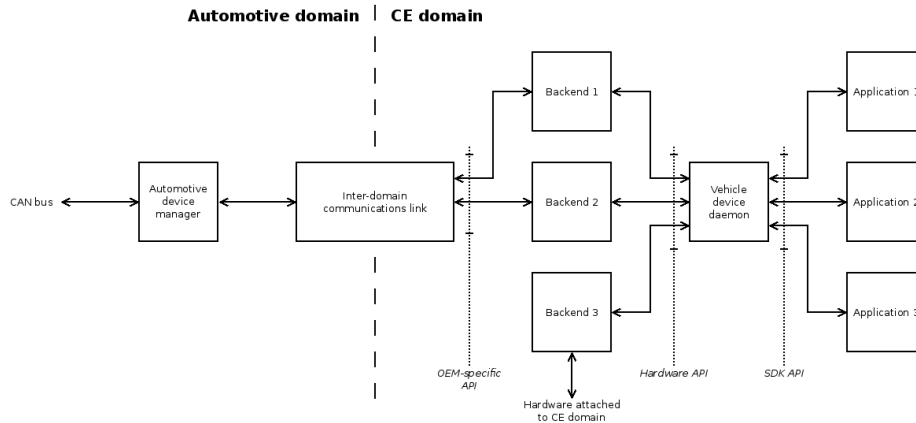
678 Approach

679 Based on the above research (**Existing sensor systems**) and **Requirements**, we
680 recommend the following approach as an initial sketch of a Sensors and Actua-
681 tors API.

²⁴<https://openconnectivity.org/technology/reference-implementation/alljoyn/>

²⁵<https://openconnectivity.org/>

682 **Overall architecture**



683

684 **Vehicle device daemon**

685 Implement a vehicle device daemon which aggregates all sensor data in the vehi-
686 cle, and multiplexes access to all actuators in the vehicle (apart from specialised
687 high bandwidth devices; see **High bandwidth or low latency sensors**). It will
688 connect to whichever underlying buses are used by the OEM to connect devices
689 (for example, the CAN and LIN buses); see **Hardware and app APIs**. The im-
690 plementation may be new, or may be a modified version of ambd, although it
691 would need large amounts of rework to fit the Apertis design (see **Automotive
692 message broker**).

693 The daemon needs to receive and process input within the latency bounds of
694 the sensors.

695 The daemon should expose a D-Bus interface which follows the W3C **Vehicle
696 Information Access API**²⁶. The set of supported properties, out of those defined
697 by the **Vehicle Signal Specification**²⁷, may vary between vehicles —this is as ex-
698 pected by the specification. It may vary over time as devices dynamically appear
699 and disappear, which programs can monitor using the **Availability interface**²⁸.

700 The W3C specification was chosen rather than something like HomeKit due to
701 its close match with the requirements, its automotive background, and the fact
702 that it looks like an active and supported specification. Furthermore, HomeKit
703 requires each device to define one or more protocols to use, allowing for arbitrary
704 flexibility in how devices communicate with the controller. All the sensor and
705 actuator use cases which are relevant to vehicles need only a property interface,
706 however, which supports getting and setting properties, and being notified when
707 they change.

²⁶http://www.w3.org/2014/automotive/vehicle_spec.html

²⁷https://github.com/COVESA/vehicle_signal_specification

²⁸http://www.w3.org/2014/automotive/vehicle_spec.html#data-availability

708 If an OEM, third party or application developer wishes to add new sensor or
709 actuator types, they should follow the [extension process](#)²⁹ and request that the
710 extensions be standardised by Apertis —they will then be released in the next
711 version of the Sensors and Actuators API, available for all applications to use.
712 If a vehicle needs to be released with those sensors or actuators in the meantime,
713 their properties must be added to the SDK API in an OEM-specific namespace.
714 Applications from the OEM can use properties from this namespace until they
715 are standardised in Apertis. See [Property naming](#).

716 Multiple vehicles can be supported by exposing new top-level instances of the
717 [Vehicle interface](#)³⁰. For example, each vehicle could be exposed as a new object
718 in D-Bus, each implementing the Vehicle interface, with changes to the set of
719 vehicles notified using an interface like the standard [D-Bus ObjectManager](#)³¹
720 interface.

721 This API can be exposed to application bundles in any binding language sup-
722 ported by GObject Introspection (including JavaScript), through the use of a
723 client library, just as with other Apertis services. The client library may pro-
724 vide more specific interfaces than the D-Bus interface —the D-Bus API may be
725 defined in terms of string keywords and variant values, whereas the client library
726 API may be sensor-specific strongly typed interfaces.

727 **Hardware and app APIs**

728 The vehicle device daemon will have two APIs: the D-Bus SDK API exposed
729 to applications, and the hardware API it consumes to provide access to the
730 CAN and LIN buses (for example). The SDK API is specified by Apertis, and
731 is standardised across all Apertis deployments in vehicles, so that a bundle
732 written against it will work in all vehicles (subject to the availability of the
733 devices whose properties it uses).

734 **Open question:** The exact definition of the SDK API is yet to be finalised. It
735 should include support for accessing multiple properties in a single IPC round
736 trip, to reduce IPC overheads.

737 The hardware API is also specified by Apertis, and implemented by one or more
738 backend services which connect to the vehicle buses and devices and expose the
739 information as properties understandable by the vehicle device daemon, using
740 the hardware API.

741 At least one backend service must be provided by the vehicle OEM, and it
742 must expose properties from the vehicle’s standard devices from the vehicle
743 buses. Other backend services may be provided by the vehicle OEM for other
744 devices, such as optional devices for premium vehicle models; or truck installa-
745 tions. Similarly, backend services may be provided by third parties for other

²⁹https://genivi.github.io/vehicle_signal_specification/rule_set/private_branch/

³⁰<https://www.w3.org/Submission/vsso/#Vehicle>

³¹<http://dbus.freedesktop.org/doc/dbus-specification.html#standard-interfaces-objectmanager>

746 devices, such as after-market devices like roof boxes. Application bundles may
747 provide backend services as well, to expose hardware via application-specific
748 protocols. Consequently, backend services will likely be developed in isolation
749 from each other.

750 Each backend service must expose zero or more properties —it is possible for
751 a backend to expose zero properties if the device it targets is not currently
752 connected, for example.

753 Each backend service must run as a separate process, communicating with the
754 vehicle device daemon over D-Bus using the hardware API. The hardware API
755 needs the following functionality:

- 756 • Bulk enumeration of vehicles
- 757 • Bulk notification of changes to vehicle availability
- 758 • Bulk enumeration of properties of a vehicle, including readability and
759 writability
- 760 • Bulk notification of changes to property availability, readability or
761 writability
- 762 • Subscription to and unsubscription from property change notifications
- 763 • Bulk property change notifications for subscribed properties

764 The hardware API will be roughly a similar shape to the SDK API, and hence
765 a lot of complexity of the vehicle device daemon will be in the vehicle-specific
766 backends (both operate on properties —**Properties vs devices**).

767 As vehicle networks differ, the backend used in a given vehicle has to be de-
768 veloped by the OEM developing that vehicle. Apertis may be able to provide
769 some common utility functions to help in implementing backends, but cannot
770 abstract all the differences between vehicles. (See **Background on intra-vehicle**
771 **networks**).

772 It is expected that the main backend service for a vehicle, provided by that ve-
773 hicle’s OEM, will be access the vehicle-specific network implementation running
774 in the automotive domain, and hence will use the **inter-domain communications**
775 **connection**³². In order to avoid additional unnecessary inter-process communi-
776 cation (IPC) hops, it is suggested that the main backend service acts as *the*
777 proxy for sensor data on the inter-domain connection, rather than communicat-
778 ing with a separate proxy in the CE domain —but only if this is possible within
779 the security requirements on inter-domain connection proxies.

780 The path for a property to pass from a hardware sensor through to an application
781 is long: from the hardware sensor, to the backend service, through the D-Bus
782 daemon to the vehicle device daemon, then through the D-Bus daemon again

³²<https://www.apertis.org/concepts/archive/application/inter-domain-communication/>

783 to the application. This is at least 5 IPC hops, which could introduce non-
784 negligible latency. See [High bandwidth or low latency sensors](#) for discussion
785 about this.

786 **Interactions between backend services** In order to keep the security
787 model for the system simple, backend services must not be able to interact.
788 Each device must be exposed by exactly one backend service —two backend
789 services cannot expose the same device; and neither can they extend devices
790 exposed by other backend services.

791 The vehicle device daemon must aggregate the properties exposed by its back-
792 ends and choose how to merge them. For example, if one backend service
793 provides a 'lights'property as an array with one element, and another backend
794 service does similarly, the vehicle device daemon should append the two and
795 expose a 'lights'array with both elements in the SDK API.

796 For other properties, the vehicle device daemon should combine scalar values.
797 For example, if one backend service exposes a rain sensor measurement of 4/10,
798 and another exposes a second measurement (from a separate sensor) of 6/10,
799 the SDK API should expose an aggregated rain sensor measurement of (for
800 example) 6/10 as the maximum of the two.

801 **Open question:** The exact means for aggregating each property in the Vehicle
802 Signal Specification is yet to be determined.

803 **Recommended hardware API design** Below is a pseudo-code recommen-
804 dation for the hardware API. It is not final, but indicates the current best
805 suggestion for the API. It has two parts —a management API which is imple-
806 mented by the vehicle device daemon; and a property API which is implemented
807 by each backend service and queried by the vehicle device daemon.

808 Types are given in the [D-Bus type system notation](#)³³.

809 **Management API** Exposed on the well-known name `org.apertis.Rhodydd1`
810 from the main daemon, the `/org/apertis/Rhodydd1` object implements the stan-
811 dard `org.freedesktop.DBus.ObjectManager`³⁴ interface to let client discover and
812 get notified about the registered vehicles.

813 Vehicles are mapped under `/org/apertis/Rhodydd1/${vehicle_id}` and implement
814 the `org.apertis.Rhodydd1.Vehicle` interface:

```
815 interface org.apertis.Rhodydd1.Vehicle {  
816     readonly property s VehicleId;  
817     method GetAttributes (  
818         in s node_path,
```

³³<http://dbus.freedesktop.org/doc/dbus-specification.html#type-system>

³⁴<http://dbus.freedesktop.org/doc/dbus-specification.html#standard-interfaces-objectmanager>


```

819     out x current_time,
820     out a(s(vdx)a{sv}(uu)) attributes)
821 method GetAttributesMetadata (
822     in s node_path,
823     out x current_time,
824     out a(sa{sv}(uu)) attributes_metadata)
825 method SetAttributes (
826     in a{sv} attributes_value)
827 method UpdateSubscriptions (
828     in a(sa{sv}) subscriptions,
829     in a(sa{sv}) unsubscriptions)
830 signal AttributesChanged (
831     x current_time,
832     a(s(vdx)a{sv}(uu)) changed_attributes,
833     a(sa{sv}(uu)) invalidated_attributes))
834 signal AttributesMetadataChanged (
835     x current_time,
836     a(sa{sv}(uu)) changed_attributes_metadata)
837 }

```

838 Backends register themselves on the bus with well-known names under the
839 `org.apertis.Rhodydd1.Backends.` prefix and implement the same interfaces and
840 the main daemon, which will monitor the owned names on the bus and register
841 to the object manager signals to multiplex access to the backends.

842 Each attribute managed via the vehicle attribute API is identified by a prop-
843 erty name. Properties names come from the Vehicle Signal Specification, for
844 example:

- 845 • [Sunroof.Position](#)³⁵
- 846 • [Horn.IsActive](#)³⁶
- 847 • `Seat.FancySeatController.BackTemperature` (oem specific property)

848 Each attribute has three values associated:

- 849 • its value (of type v)
- 850 • its accuracy (as a standard deviation of type d, set to 0.0 for non-numeric
851 values)
- 852 • the timestamp when it was last updated (of type x)

853 In addition the current time is also returned for comparison to the time the
854 value was last updated.

855 Values also have two set of metadata (of type u) associated:

- 856 • availability enum

³⁵<https://www.w3.org/Submission/vsso/#SunroofPositionSensor>

³⁶<https://www.w3.org/Submission/vsso/#HornIsActive>

857 – AVAILABLE = 1
858 – NOT_SUPPORTED = 0
859 – NOT_SUPPORTED_YET = 2
860 – NOT_SUPPORTED_SECURITY_POLICY = 3
861 – NOT_SUPPORTED_BUSINESS_POLICY = 4
862 – NOT_SUPPORTED_OTHER = 5
863 • access flags
864 – NONE = 0
865 – READABLE = (1 « 0)
866 – WRITABLE = (1 « 1)

867 The GetAttributes method must return the value of all properties in the given
868 branch indicated by the node path. If the node path represents a leaf node, then
869 only the value corresponding to that property is returned. If no such branch or
870 property exists on that vehicle, it must return an error. To get all properties of
871 the vehicle an empty node path shall be passed.

872 To receive notification of attribute changes via the AttributesChanged and At-
873 tributesMetadataChanged signals, clients must first register their subscription
874 with the UpdateSubscriptions method to specify the kind of properties for which
875 they have some interest.

876 A backend service must emit an AttributesChanged signal when one of the
877 properties it exposes changes, but it may wait to combine that signal with those
878 from other changed properties —the trade-off between latency and notification
879 frequency should be determined by backend service developers.

880 **Hardware API compliance testing**

881 As the vehicle-specific and third party backend services to the vehicle device
882 daemon contain a large part of the implementation of this system, there should
883 be a compliance test suite which all backend services must pass before being
884 deployed in a vehicle.

885 If a backend service is provided by an application bundle, that application bun-
886 dle must additionally undergo more stringent app store validation, potentially
887 including a requirement for security review of its code. See [Checks for backend](#)
888 [services](#).

889 The compliance test suite must be automated, and should include a variety of
890 tests to ensure that the hardware API is used correctly by the backend service.
891 It should be implemented as a mock D-Bus service which mocks up the hardware
892 management API ([Recommended hardware API design](#)), and which calls the
893 hardware property API. The backend service must be run against this mock
894 service, and call its methods as normal. The mock service should return each
895 of the possible return values for each method, including:

- 896 • Success.

- 897 • Each failure code.
- 898 • Timeouts.
- 899 • Values which are out of range.

900 It must call property API methods with various valid and invalid input.

901 The backend service must not crash or obviously misbehave (such as consuming
902 an unexpected amount of CPU time or memory).

903 As the backend service pushes data to the vehicle device daemon, the compliance
904 test could be trivially passed by a backend service which pushes zero properties
905 to it. This must not be allowed: backend services must be run under a test
906 harness which triggers all of their behaviour, for all of the devices they support.
907 Whether this harness simulates traffic on an underlying intra-vehicle network,
908 or physically provides inputs to a hardware sensor, is implementation defined.
909 The behaviour must be consistently reproducible for multiple compliance test
910 runs.

911 **SDK API compliance testing and simulation**

912 Application bundle developers will not be able to test their bundles on real
913 vehicles easily, so a simulator should be made available as part of the SDK, which
914 exposes a developer-configurable set of properties to the bundle under test. The
915 simulator must support all properties and configurations supported by the real
916 vehicle device daemon, including multiple vehicles and third-party accessories;
917 otherwise bundles will likely never be tested in such configurations. Similarly,
918 it must support varying properties over time, simulating dynamic addition and
919 removal of vehicles and devices, and simulating errors in controlling actuators
920 (for example, [Automatic window feedback](#)).

921 The emulator should be implemented as a special backend service for the vehicle
922 device daemon, which is provided by the emulator application. That way, it can
923 directly feed simulated device properties into the daemon. This backend, and
924 the emulator should only be available on the SDK, and must never be available
925 on production systems.

926 Compliance testing of application bundles is harder, but as a general principle,
927 any of the [Apertis store validation](#) checks which *can* be brought forward so they
928 can be run by the bundle developers, *should* be brought forward.

929 **SDK hardware**

930 If a developer has appropriate sensors or actuators attached to their development
931 machine, the development version of the sensors and actuators system should
932 have a separate backend service which exposes that hardware to applications
933 for development and testing, just as if it were real hardware in a vehicle.

934 This backend service must be separate from the emulator backend service (
935 **SDK API compliance testing and simulation**), in order to allow them to be used
936 independently.

937 **Trip logging of sensor data**

938 As well as an emulator for application developers to use when testing their
939 applications, it would be useful to provide pre-recorded ‘trip logs’ of sensor data
940 for typical driving trips which an application should be tested against. These
941 trip logs should be replayable in order to test applications.

942 The design for this is covered in the ‘Trip logging of SDK sensor data’ section of
943 the Debug and Logging design.

944 **Properties vs devices**

945 A major design decision was whether to expose individual sensors to bundles
946 via the SDK API, or to expose properties of the vehicle, which may correspond
947 to the reading from a single sensor or to the aggregate of readings from multiple
948 sensors. For example, if exposing sensors, the API would expose a gyroscope
949 plus several accelerometers, each returning individual one-dimensional measure-
950 ments. Bundles would have to process and aggregate this data themselves—in
951 the majority of cases, that would lead to duplication of code (and most likely
952 to bugs in applications where they mis-process the data), but it would also
953 allow more advanced bundles access to the raw data to do interesting things
954 with. Conversely, if exposing properties, the vehicle device daemon would pre-
955 aggregate the data so that the properties exposed to bundles are filtered and
956 averaged acceleration values in three dimensions and three angular dimensions.
957 This would simplify implementation within bundles, at the cost of preventing a
958 small class of interesting bundles from accessing the raw data they need.

959 For the sake of keeping bundles simpler, and hence with potentially fewer bugs,
960 this design exposes properties rather than sensors in the SDK API. This also
961 means that the potentially latency sensitive aggregation code happens in the
962 daemon, rather than in bundles which receive the data over D-Bus, which has
963 variable latency.

964 Similarly, the hardware API must expose properties as well, rather than indi-
965 vidual devices. It may aggregate data where appropriate (for example, if it has
966 information which is useful to the aggregation process which it cannot pass on
967 to the vehicle device daemon). This also means that a set of device semantics,
968 separate from the W3C Vehicle Data property semantics, does not have to be
969 defined; nor a mapping between it and the properties.

970 **Property naming**

971 Properties exposed in the SDK API must be named following the Vehicle Signal
972 Specification (VSS) [naming guidelines](#)³⁷. VSS defines a 'tree-like' logical taxon-
973 omy of the vehicle, (formally a Directed Acyclic Graph), where major vehicle
974 structures (e.g. body, engine) are near the top of the tree and the logical assem-
975 blies and components that comprise them, are defined as their child nodes. Each
976 of the child nodes in the tree is further decomposed into its logical constituents,
977 and the process is repeated until leaf nodes are reached. A leaf node is a node
978 at the end of a branch that cannot be decomposed because it represents a single
979 signal or data attribute value. For example some of the properties of DriveTrain
980 transmission and fuel system are exposed with these names:

- 981 • [Drivetrain.Transmission.Speed](#)³⁸
- 982 • [Drivetrain.Transmission.TravelledDistance](#)³⁹
- 983 • [DriveTrain.FuelSystem.TankCapacity](#)⁴⁰

984 The element hops from the root to the leaf is called path. Properties are named
985 according to their path from the root of the tree toward the node itself and each
986 element in the path is delimited by using the dot notation.

987 Property names are formed of components in the data tree (which may contain
988 the letters a-z, A-Z, and the digits 0-9; they must start with a letter a-z or A-Z,
989 and must be in CamelCase) separated by dots. Property names must start and
990 end with a component (not a dot) and contain one or more components.

991 If an OEM needs to expose a custom (non-standardised) property, they must
992 define them underneath the [private branch](#)⁴¹ which is provided by VSS to facil-
993 itate OEM specific properties.

994 **High bandwidth or low latency sensors**

995 Sensors which provide high bandwidth outputs, or whose outputs must reach the
996 bundle within certain latency bounds (as opposed to simply being aggregated
997 by the vehicle device daemon within certain latency bounds), will be handled
998 out of band. Instead of exposing the sensor data via the vehicle device daemon,
999 the address of some out of band communications channel will be exposed. For
1000 video devices, this might be a V4L device node; for audio devices it might be a
1001 PulseAudio device identifier. Multiplexing access to the device is then delegated
1002 to the out of band mechanism.

1003 This considerably relaxes the performance requirements on the vehicle device

³⁷https://covesa.github.io/vehicle_signal_specification/rule_set/basics/#addressing-nodes

³⁸<https://www.w3.org/Submission/vsso/#VehicleSpeed>

³⁹<https://www.w3.org/Submission/vsso/#TravelledDistance>

⁴⁰<https://www.w3.org/Submission/vsso/#tankCapacity>

⁴¹https://genivi.github.io/vehicle_signal_specification/rule_set/private_branch/

1004 daemon, and allows the more specialist high bandwidth use cases to be handled
1005 by more specialised code designed for the purpose.

1006 **Timestamps and uncertainty bounds**

1007 The W3C Vehicle Signal Specification does not define uncertainty fields for
1008 any of its data types (for example, [VehicleSpeed](#)⁴² contains a single speed field
1009 measured in kilometres per hour). However, it allows the extensibility, so the
1010 data types exposed by the vehicle device daemon should all include an extension
1011 field specifying the uncertainty (accuracy) of the measurement, in appropriate
1012 units; and another specifying the timestamp when the measurement was taken,
1013 in monotonic time (in the [CLOCK_MONOTONIC](#)⁴³ sense).

1014 For example, the Apertis VehicleSpeed update looks like this:

```
1015 [ ('Drivetrain.Transmission.Speed',           -> property name  
1016     (110, 0.3, 38003116),                               -  
1017 > value field (speed, uncertainty, timestamp)  
1018   {'description': 'Lateral vehicle acceleration', -> metadata  
1019     'id': 54,  
1020     'type': 'Int32',  
1021     'unit': 'km/h'})  
1022 ]
```

1023 which represents a measurement of *speed* \pm *uncertainty* (110 \pm 0.3) kilometres
1024 per hour.

1025 **Registering triggers and actions**

1026 When subscribing to notifications for changes to a particular property using the
1027 [VehicleSignalInterface](#)⁴⁴ interface, a program is also subscribing to be woken up
1028 when that property changes, even if the program is suspended or otherwise not
1029 in the foreground.

1030 Once woken up, the program can process the updated property value, and poten-
1031 tially send a notification to the user. If the user interacts with this notification,
1032 the program may be brought to the foreground. The program must not be au-
1033 tomatically brought to the foreground without user interaction or it will steal
1034 the user's focus, which is distracting.

1035 See the draft compositor security design

1036 Alternatively, the program could process the updated property value in the
1037 background without notifying the user.

⁴²https://covesa.github.io/vehicle_signal_specification/rule_set/data_entry/sensor_uator/

⁴³https://manpages.debian.org/unstable/manpages-dev/clock_gettime.2.en.html

⁴⁴http://www.w3.org/2014/automotive/vehicle_spec.html#widl-VehicleSignalInterface-subscribe-unsigned-short-VehicleInterfaceCallback-callback-Zone-zone

1038 The VehicleSignalInterface interface may be extended to support notifications
1039 only when a property value is in a given range; a degenerate case of this, where
1040 the upper and lower bounds of the range are equal, would support notifica-
1041 tions for property values crossing a threshold. This would most likely be imple-
1042 mented by adding optional min and max parameters to the VehicleSignalInter-
1043 face.subscribe() method.

1044 **Bulk recording of sensor data**

1045 This is a slightly niche use case for the moment, and can be handled by an
1046 application bundle running an agent process which is subscribed to the relevant
1047 properties and records them itself. This is less efficient than having the vehicle
1048 device daemon do it, as it means more processes waking up for changes in sensor
1049 data, but avoids questions of data formats to use and how and when to send bulk
1050 data between the vehicle device daemon and the application bundle's agent.

1051 If the implementation of this is moved into the vehicle device daemon, the
1052 lifecycle of recorded data must be considered: how space is allocated for the
1053 data's storage, when and how the application bundle is woken to process the
1054 data, and what happens when the allocated storage space is filled.

1055 **Security**

1056 The vehicle device daemon acts as a privilege boundary between all bundles
1057 accessing devices, between the bundles and the devices, and between each back-
1058 end service. Application bundles must request permissions to access sensor data
1059 in their manifest (see the Applications Design document), and must separately
1060 request permissions to interact with actuators. The split is because being able
1061 to control devices in the vehicle is more invasive than passively reading from
1062 sensors —it is safety critical. A sensible security policy may be to further split
1063 out the permissions in the manifest to require specific permissions for certain
1064 types of sensors, such as cabin audio sensors or parking cameras, which have
1065 the potential to be used for tracking the user. As adding more permissions
1066 has a very low cost, the recommendation is to err on the side of finer-grained
1067 permissions.

1068 The manifest should additionally separate lists of device properties which the
1069 bundle *requires* access to from device properties which it *may* access if they
1070 exist. This will allow the Apertis store to hide bundles which require devices
1071 not supported by the user's vehicle.

1072 From the permissions in the manifest, AppArmor and polkit rules restricting
1073 the program's access to the vehicle device daemon's API can be generated on
1074 installation of the bundle. See [Security domains](#) for rationale.

1075 When interacting with the vehicle device daemon, a program is securely identi-
1076 fied by its D-Bus connection credentials, which can be linked back to its manifest
1077 —the vehicle device daemon can therefore check which permissions the program'

1078 s bundle holds and accept or reject its access request as appropriate. Therefore,
1079 the vehicle device daemon acts as ‘the underlying operating system’ in controlling
1080 access, in the phrasing used by⁴⁵ the W3C specification. It enforces the security
1081 boundary between each bundle accessing devices, and between the intra- and
1082 inter-vehicle networks. The vehicle device daemon forms a separate security
1083 domain from any of the applications.

1084 Each backend service is a separate security domain, meaning that the vehicle
1085 device daemon is in a separate security domain from the intra-vehicle networks.

1086 The daemon may rate-limit API requests from each program in order to prevent
1087 one program monopolising the daemon’s process time and effectively causing a
1088 denial of service to other bundles by making API calls at a high rate. This
1089 could result from badly implemented programs which poll sensors rather than
1090 subscribing to change notifications from them, for example; as well as malicious
1091 bundles.

1092 Due to its complexity, low level in the operating system, and safety criticality,
1093 the vehicle device daemon requires careful implementation and auditing by an
1094 experienced developer with knowledge of secure software development at the
1095 operating system level and experience with relevant technologies (polkit, Ap-
1096 pArmor, D-Bus).

1097 The threat model under consideration is that of a malicious or compromised
1098 bundle which can execute any of the D-Bus SDK APIs exposed by the daemon,
1099 with full manifest privileges for sensor access. A second threat model is that of
1100 a compromised backend service, which can execute any of the D-Bus hardware
1101 APIs exposed by the daemon.

1102 **Security domains** There are various security technologies available in Aper-
1103 tis for use in restricting access to sensors and actuators. See the Security Design
1104 for background on them; especially §9, Protecting the driver assistance system
1105 from attacks. These technologies can only be used on the boundaries between
1106 security domains. In this design, each application bundle is a single security
1107 domain (encompassing all programs in the bundle, including agents and helper
1108 programs); the vehicle device daemon is another domain; and each of the back-
1109 end services are in a separate domain (including the vehicle networks they each
1110 use).

1111 **Application bundle and another application bundle or the rest of the**
1112 **system** Separation of the security domains of different application bundles
1113 from each other and from the rest of the system is covered in the Applications
1114 and Security designs.

1115 **Application bundle and vehicle device daemon** The boundary between
1116 an application bundle and the vehicle device daemon is the Sensors and Actu-

⁴⁵http://www.w3.org/2014/automotive/vehicle_spec.html#security

1117 ators SDK API, implemented by the daemon and exposed over D-Bus. The
1118 bundle's AppArmor profile will grant access to call any method on this interface
1119 if and only if the bundle requests access to one or more devices in its manifest.
1120 Note that AppArmor is not used to separate access to different sensors or actua-
1121 tors —it is not fine-grained enough, and is limited to allowing or denying access
1122 to the API as a whole.

1123 A separate set of [polkit](http://www.freedesktop.org/software/polkit/docs/master/polkit.8.html)⁴⁶ rules for the bundle control which devices the bundle is
1124 allowed to access; these rules are generated from the bundle's manifest, looking
1125 at the specific devices listed. Given a set of polkit actions defined by the vehicle
1126 device daemon, these rules should permit those actions for the bundle.

1127 For example, the daemon could define the polkit actions:

- 1128 • org.apertis.vehicle_device_daemon.EnumerateVehicles: To list the avail-
1129 able vehicles or subscribe to notifications of changes in the list.
- 1130 • org.apertis.vehicle_device_daemon.EnumerateDevices: To list the avail-
1131 able devices on a given vehicle (passed as the vehicle variable on the action)
1132 or subscribe to notifications of changes in the list.
- 1133 • org.apertis.vehicle_device_daemon.ReadProperty: To read a property, i.e.
1134 access a sensor, or subscribe to notifications of changes to the property
1135 value. The vehicle ID and property name are passed as the vehicle and
1136 property variables on the action.
- 1137 • org.apertis.vehicle_device_daemon.WriteProperty: To write a property,
1138 i.e. operate an actuator. The vehicle ID, property name and new value
1139 are passed as the vehicle, property and value variables on the action.

1140 The default rules for all of these actions must be `polkit.Result.NO`.

1141 If a bundle has access to any device, it is safe and necessary to grant it access to
1142 enumerate *all* vehicles and devices (the Enumerate* actions above) —otherwise
1143 the bundle cannot check for the presence of the devices it requires. Knowledge
1144 of which devices are connected to the vehicle should not be especially sensitive
1145 —it is expected that there will not be a sufficient variety of devices connected
1146 to a single vehicle to allow fingerprinting of the vehicle from the device list, for
1147 example.

1148 An application bundle, `org.example.AccelerateMyMirror`, which requests
1149 access to the `vehicle.throttlePosition.value` property (a sensor) and the `vehic-
1150 cle.mirror.mirrorPan` property (an actuator) would therefore have the following
1151 polkit rule generated in `/etc/polkit-1/rules.d/20-org.example.AccelerateMyMirror.rules`:

```
1152 polkit.addRule (function (action, subject) {  
1153     if (subject.credentials != 'org.example.AccelerateMyMirror') {  
1154         /* This rule only applies to this bundle.  
1155         * Defer to other rules to handle other bundles. */
```

⁴⁶<http://www.freedesktop.org/software/polkit/docs/master/polkit.8.html>

```

1156     return polkit.Result.NOT_HANDLED;
1157 }
1158
1159 if (action.id == 'org.apertis.vehicle_device_daemon.EnumerateVehicles' ||
1160     action.id == 'org.apertis.vehicle_device_daemon.EnumerateDevices') {
1161     /* Always allow these. */
1162     return polkit.Result.YES;
1163 }
1164
1165 if (action.id == 'org.apertis.vehicle_device_daemon.ReadProperty' &&
1166     action.lookup ('property') == 'vehicle.throttlePosition.value') {
1167     /* Allow access to this specific property. */
1168     return polkit.Result.YES;
1169 }
1170
1171 if (action.id == 'org.apertis.vehicle_device_daemon.WriteProperty' &&
1172     action.lookup ('property') == 'vehicle.mirror.mirrorPan') {
1173     /* Allow access to this specific property,
1174      * with user authentication. */
1175     return polkit.Result.AUTH_USER;
1176 }
1177
1178 /* Deny all other accesses. */
1179 return polkit.Result.NO;
1180 });

```

1181 In the rules, the subject is always the program in the bundle which is requesting
1182 access to the device.

1183 **Open question:** What is the exact security policy to implement regarding
1184 separation of sensors and actuators? For example, bundle access to sensors
1185 could always be permitted without prompting by returning `polkit.Result.YES`
1186 for all sensor accesses; but actuator accesses could always be prompted to the
1187 user by returning `polkit.Result.AUTH_SELF`. The choice here depends on the
1188 desired user experience.

1189 **Vehicle device daemon and a backend service** The boundary between
1190 the vehicle device daemon and one of its backend services is the Sensors and
1191 Actuators hardware API, implemented by the daemon and exposed over D-Bus.
1192 The backend service's AppArmor profile will grant access to call any method on
1193 this interface. Note that AppArmor is not used to grant or deny permissions
1194 to expose particular properties —it is not fine-grained enough, and is limited to
1195 allowing or denying access to the API as a whole.

1196 In order to limit the potential for a compromised backend service to escalate its
1197 compromise into providing malicious sensor data for any sensor on the system,
1198 each backend service must install a file which lists the Vehicle Data properties

1199 it might possibly ever provide to the vehicle device daemon. The vehicle device
1200 daemon must reject properties from a backend service which are not in this list.
1201 The list must not be modifiable by the backend service after installation (i.e. it
1202 must be read-only, readable by the vehicle device daemon).

1203 Furthermore, if a backend service is found to be exploitable after being deployed,
1204 it must be possible for the vehicle device daemon to disable it. This is expected
1205 to typically happen with backend services provided by application bundles, as
1206 opposed to those provided by OEMs or third parties (as these should go through
1207 stricter review, and disabling them would have a much larger impact). The
1208 vehicle device daemon must have a blacklist of backend services which it never
1209 loads. It must check the credentials of D-Bus messages from backend services
1210 against this blacklist.

1211 Using `GetConnectionCredentials`, which returns an unforgeable iden-
1212 tifier for the peer: [http://dbus.freedesktop.org/doc/dbus-specificat](http://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-get-connection-credentials)
1213 [ion.html#bus-messages-get-connection-credentials](http://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-get-connection-credentials)

1214 In order to support one (vulnerable) version of a backend service being black-
1215 listed, but not the next (fixed) version, the blacklist must contain version num-
1216 bers, which should be compared against the installed version number of the
1217 backend service as listed in the system-wide application bundle manifest store.

1218 **Vehicle device daemon and the rest of the system** The vehicle device
1219 daemon itself must not be able to access any of the vehicle buses or any networks.
1220 It must be run as a unique user, which owns the daemon's binary, with its DAC
1221 permissions set such that other users (except root) cannot run it. It must not
1222 have access to any device files. See §9, Protecting the driver assistance system
1223 from attacks, of the Security design for more details.

1224 **Backend service and another backend service or the rest of the system**
1225 In order to guarantee it is the only program which can access a particular vehicle
1226 bus or network, each backend service should run as a unique user. The service'
1227 s binary must be owned by that user, with its DAC permissions set such that
1228 other users (except root) cannot run it. Any device files which it uses for access
1229 to the underlying vehicle networks must be owned by that user, with their DAC
1230 permissions set such that other users cannot access them, and udev rules in place
1231 to prevent access by other users. If the backend needs access to a (local) network
1232 interface to communicate with the vehicle network buses, that interface must
1233 be put in a separate network namespace, and the `CLONE_NEWNET` flag used
1234 when spawning the backend service to put it in that namespace. This prevents
1235 the service from accessing other network interfaces; and prevents other processes
1236 from accessing the buses. See §9, Protecting the driver assistance system from
1237 attacks, of the Security design for more details.

1238 **SDK emulator** Typically, it should not be possible for one program to have
1239 access to both the vehicle device daemon's SDK API and its hardware API (this

1240 access is controlled by AppArmor). However, the SDK emulator is a special case
1241 which needs access to both —so either this must be possible as a special case, or
1242 the SDK emulator must be split into a backend service process and a UI process,
1243 which communicate via another D-Bus connection.

1244 **Apertis store validation** Application bundles which request permissions to
1245 access devices must undergo additional checks before being put on the Apertis
1246 store. This is especially important for bundles which request access to actuators,
1247 as those bundles are then potentially safety critical.

1248 **Checks for access to sensors** Suggested checks for bundles requesting read
1249 access to sensors:

- 1250 • The bundle does not send privacy-sensitive data to services outside the
1251 user’s control (for example, servers not operated by the user; see the [User
1252 Data Manifesto⁴⁷](#)), either via network transmission, logging to local stor-
1253 age, or other means, without the user’s consent. Any data sent *with* the
1254 user’s consent must only be sent to services which follow the User Data
1255 Manifesto. For example (this list is not exhaustive):
 - 1256 – Tracking the vehicle’s movements.
 - 1257 – Monitoring the user’s conversations (audio recording).
- 1258 • The bundle does not have access to uniquely identifiable information, such
1259 as a vehicle identification number (VIN). Any exceptions to this would
1260 need stricter review.
- 1261 • The bundle clearly indicates when it is gathering privacy-sensitive data
1262 from sensors. For example, a ‘recording’light displayed in the UI when
1263 listening using a microphone.
 - 1264 1.

1265 Suggested checks for bundles requesting write access to actuators:

- 1266
- 1267 • The bundle does not additionally have network access.
- 1268 • Actuators are only operated while the vehicle is not driving. Any excep-
1269 tions to this would need even stricter review.
- 1270 • Manual code review of the entire bundle’s source code by a developer with
1271 security experience. The entire source code must be made available for
1272 review by the bundle developer, as it is all run in the same security domain.
1273 For example (this list is not exhaustive):

1274

⁴⁷<https://userdatamanifesto.org/>

1275 – Looking for ways the bundle could potentially be exploited by an
1276 attacker.

1277 – Checking that the bundle cannot use the actuator inappropriately
1278 during normal operation if it encounters unexpected circumstances.
1279 (For example, checking that arithmetic bugs don't exist which could
1280 cause an actuator to be operated at a greater magnitude than in-
1281 tended by the bundle developer.)

1282 **Open question:** The specific set of Apertis store validation checks for bundles
1283 which access devices is yet to be finalised.

1284

1285 **Checks for backend services** Suggested checks for backend services for the
1286 vehicle device daemon, whether they are provided by an OEM, a third party or
1287 as part of an application bundle:

- 1288 • The backend service does not additionally have network access.
- 1289 • The backend service does not have write access to any of the file system
1290 except devices it needs, and the D-Bus socket.
- 1291 • The backend service cannot access any more device nodes than it needs
1292 to support its devices.
- 1293 • Manual code review of the entire bundle's source code by a developer with
1294 security experience. The entire source code must be made available for
1295 review by the bundle developer, as it is all run in the same security domain.
1296 For example (this list is not exhaustive):
 - 1297 – Looking for ways the backend service could potentially be exploited
1298 by an attacker.
 - 1299 – Checking that the backend service cannot use any of its actuator
1300 inappropriately during normal operation if it encounters unexpected
1301 circumstances. (For example, checking that arithmetic bugs don'
1302 t exist which could cause an actuator to be operated at a greater
1303 magnitude than intended by the developer.)
- 1304 • The backend service's D-Bus service is only accessible by the vehicle device
1305 daemon (as enforced by AppArmor).
- 1306 • If other software is shipped in the same application bundle, it must be
1307 considered to be part of the same security domain as the backend service,
1308 and hence subject to the same validation checks.
- 1309 • The backend service must pass the automated compliance test ([Hardware
1310 API compliance testing](#)).
- 1311 • The backend service must not expose any properties which are not sup-
1312 ported by the version of the vehicle device daemon which it targets as its

1313 minimum dependency (see [Vehicle device daemon](#) for information about
1314 the extension process).

1315 **Suggested roadmap**

1316 Due to the large amount of work required to write a system like this from scratch,
1317 it is worth exploring whether it can be developed in stages.

1318 The most important parts to finalise early in development are the SDK and hard-
1319 ware APIs, as these need to be made available to bundle developers and OEMs
1320 to develop bundles and the backend services. There seems to be little scope for
1321 finalising these APIs in stages, either (for example by releasing property access
1322 APIs first, then adding vehicle and device enumeration), as that would result in
1323 early bundles which are incompatible with multi-vehicle configurations.

1324 Similarly, it does not seem to be possible to implement one of the APIs before
1325 the other. Due to the fragmented nature of access to vehicle networks, the
1326 backend needs to be written by the OEM, rather than relying on one written
1327 by Apertis for early versions of the system.

1328 Furthermore, the security implementation for the vehicle device daemon must
1329 be part of the initial release, as it is safety critical.

1330 One area where phased development is possible is in the set of properties itself
1331 —initial versions of the daemon and backends could implement a small, core set
1332 of the properties defined in the [VSS Ontology \(VSSo\)](#)⁴⁸, and future versions
1333 could expand that set of properties as time is available to implement them. As
1334 each property is a public API, it must be supported as part of the SDK one it
1335 has appeared in a released version of the daemon, so it is important to design
1336 the APIs correctly the first time.

1337 Similarly, the scope for backend services could be expanded over time. Initial
1338 releases of the system could allow only backend services written by vehicle OEMs
1339 to be used; with later releases allowing third-party backend services, then ones
1340 provided by installed application bundles.

1341 The emulator backend service ([SDK API compliance testing and simulation](#))
1342 and any SDK hardware backend services ([SDK hardware](#)) should be imple-
1343 mented early on in development, as they should be relatively simple, and hav-
1344 ing them allows application developers to start writing applications against the
1345 service.

1346 **Requirements**

- 1347 • [Enumeration of devices](#): The availability of known properties of the vehicle
1348 can be checked through the [Availability interface](#)⁴⁹. The W3C approach

⁴⁸<https://www.w3.org/Submission/vsso/>

⁴⁹http://www.w3.org/2014/automotive/vehicle_spec.html#data-availability

1349 considers properties, rather than devices, to be the enumerable items, but
 1350 they are mostly equivalent (see [Properties vs devices](#)).

- 1351 • **Enumeration of vehicles:** The availability of objects implementing the
 1352 W3C Vehicle interface on D-Bus is exposed using an interface like the
 1353 D-Bus ObjectManager API.
- 1354 • **Retrieving data from sensors:** Properties can be retrieved through the
 1355 [VehicleInterface interface](#)⁵⁰. For high bandwidth sensors, or those with
 1356 latency requirements for the end-to-end connection between sensor and
 1357 bundle, data is transferred out of band (see [High bandwidth or low latency](#)
 1358 [sensors](#)).
- 1359 • **Sending data to actuators:** Properties can be set through the [VehicleSig-](#)
 1360 [nalInterface](#)⁵¹ interface. As with getting properties, data for high band-
 1361 width or low latency sensors is transferred out of band.
- 1362 • **Network independence:** The vehicle device daemon abstracts access to the
 1363 underlying buses, so bundles are unaware of it.
- 1364 • **Bounded latency of processing sensor data:** The vehicle device daemon
 1365 should have its scheduling configuration set so that it can provide latency
 1366 guarantees for the underlying buses.
- 1367 • **Extensibility for OEMs:** Extensions are standardised through Apertis and
 1368 released in the next version of the Sensors and Actuators API for use by
 1369 the OEM.
- 1370 • **Third-party backends:** Backend services for the vehicle device daemon
 1371 can be installed as part of application bundles (either built-in or store
 1372 bundles).
- 1373 • **Third-party backend validation:** Backend services must be validated be-
 1374 fore being installed as bundles (see [Checks for backend services](#)).
- 1375 • **Notifications of changes to sensor data:** Property changes are notified
 1376 via a publish-subscribe interface on [VehicleSignalInterface](#)⁵². Notification
 1377 thresholds are supported by optional parameters on that interface.
- 1378 • **Uncertainty bounds:** The W3C API is extended to include uncertainty
 1379 bounds for measurements.
- 1380 • **Failure feedback:** Through its use of [Promises](#)⁵³, the API allows for failure
 1381 to set a property.

⁵⁰<https://www.w3.org/Submission/vsso/#Vehicle>

⁵¹http://www.w3.org/2014/automotive/vehicle_spec.html#widl-VehicleSignalInterface-subscribe-unsigned-short-VehicleInterfaceCallback-callback-Zone-zone

⁵²http://www.w3.org/2014/automotive/vehicle_spec.html#widl-VehicleSignalInterface-subscribe-unsigned-short-VehicleInterfaceCallback-callback-Zone-zone

⁵³<http://www.w3.org/TR/2013/WD-dom-20131107/#promises>

- 1382 • **Timestamping:** The W3C API is extended to include timestamps for mea-
1383 surements.
- 1384 • **Triggering bundle activation:** Programs are woken by subscriptions to
1385 property changes (see [Registering triggers and actions](#)).
- 1386 • **Bulk recording of sensor data:** **Not currently implemented**, but may
1387 be implemented in future as a straightforward extension to the API. See
1388 [Bulk recording of sensor data](#).
- 1389 • **Sensor security:** Access to the Sensors and Actuators API is controlled by
1390 an AppArmor profile generated from permissions in the manifest. Access
1391 to individual sensors is controlled by a polkit rule generated from the same
1392 permissions. See [Security](#).
- 1393 • **Actuator security:** As with [Sensor security](#); sensors and actuators are
1394 listed and controlled by the polkit profile separately.
- 1395 • **App-store knowledge of device requirements:** As devices required by an
1396 application bundle are listed in the bundle's manifest (see [Security](#)), the
1397 Apertis store knows whether the bundle is supported by the user's vehicle.
- 1398 • **Accessing devices on multiple vehicles:** Each vehicle is exposed as a sepa-
1399 rate D-Bus object, each implementing the W3C Vehicle interface.
- 1400 • **Third-party accessories:** Properties for third-party accessories must be
1401 standardised through Apertis and exposed as separate interfaces on the
1402 vehicle object on D-Bus.
- 1403 • **SDK hardware support:** SDK hardware should be supported through a
1404 separate development-only backend service written specifically for that
1405 hardware.

1406 Open questions

- 1407 1. **Hardware and app APIs:** The exact definition of the SDK API is yet to
1408 be finalised. It should include support for accessing multiple properties in
1409 a single IPC round trip, to reduce IPC overheads.
- 1410 2. **Interactions between backend services:** The exact means for aggregating
1411 each property in the Vehicle Data specification is yet to be determined.
- 1412 3. **Security domains:** What is the exact security policy to implement re-
1413 garding separation of sensors and actuators? For example, bundle access
1414 to sensors could always be permitted without prompting by returning
1415 polkit.Result.YES for all sensor accesses; but actuator accesses could al-
1416 ways be prompted to the user by returning polkit.Result.AUTH_SELF.
1417 The choice here depends on the desired user experience.
- 1418 4. **Apertis store validation:** The specific set of Apertis store validation checks
1419 for bundles which access devices is yet to be finalised.

1420 **Summary of recommendations**

1421 As discussed in the above sections, we recommend:

- 1422 • Implementing a vehicle device daemon which exposes the W3C Vehicle
1423 Information Access API; this will probably need to be developed from
1424 scratch.
- 1425 • Documenting the hardware API and distributing it to OEMs, third parties
1426 and application developers along with a compliance test suite and a com-
1427 mon utility library to allow them to build backend services for accessing
1428 vehicle networks.
- 1429 • Documenting the SDK API and distributing it to application bundle de-
1430 velopers along with a validation suite and simulator to allow them to build
1431 programs which use the API.
- 1432 • Provide example trip logs for journeys to test against and a method for
1433 replaying them via the vehicle device daemon, so application developers
1434 can test their applications.
- 1435 • Defining how to aggregate multiple values of each property in the W3C
1436 Vehicle Data API.
- 1437 • Extending the W3C Vehicle Information Service Specification to expose
1438 uncertainty and timestamp data for each property.
- 1439 • Extending the W3C Vehicle Information Service Specification to expose
1440 multiple vehicles and notify of changes using an interface like D-Bus Ob-
1441 jectManager.
- 1442 • Extending the W3C Vehicle Information Service Specification to support
1443 a range of interest for property change notifications.
- 1444 • Adding a property to the application bundle manifest listing which device
1445 properties programs in the bundle may access if they exist.
- 1446 • Adding a property to the application bundle manifest listing which device
1447 properties programs in the bundle require access to.
- 1448 • Extending the Apertis store validation process to include relevant checks
1449 when application bundles request permissions to access sensors (privacy
1450 sensitive) or actuators (safety critical). Or when application bundles re-
1451 quest permissions to provide a vehicle device daemon backend service
1452 (safety critical).
- 1453 • Modifying the Apertis software installer to generate AppArmor rules to
1454 allow D-Bus calls to the vehicle device daemon if device properties are
1455 listed in the application bundle manifest.
- 1456 • Modifying the Apertis software installer to generate polkit rules to grant
1457 an application bundle access to specific devices listed in the application

- 1458 bundle manifest.
- 1459 • Implementing and auditing strict DAC and MAC protection on the vehicle
1460 device daemon and each of its backend services, and identity checks on all
1461 calls between them.
 - 1462 • Defining a feedback and standardisation process for OEMs to request new
1463 properties or device types to be supported by the vehicle device daemon’s
1464 s API.

1465 Sensors and Actuators API

1466 This sections aims to compare the current status of the Vehicle device daemon
1467 for the sensors and actuators SDK API ([Rhosydd](#)⁵⁴) with the latest W3C spec-
1468 ifications: the [Vehicle Information Service Specification](#)⁵⁵ API and the [Vehicle
1469 Signal Specification](#)⁵⁶ data model.

1470 It will also explain the required changes to align Rhosydd to the new W3C
1471 specifications.

1472 Rhosydd API Current State

1473 The current Rhosydd API is stable and usable implementing the [Vehicle Infor-
1474 mation Service Specification](#)⁵⁷ and using the data model specified by the [Vehicle
1475 Signal Specification](#)⁵⁸.

1476 Considerations to align Rhosydd to the new VISS API

- 1477 1. The original Vehicle API and the Rhosydd API don’t exactly match 1:1 as
1478 the latter has been adapted to follow the inter-process D-Bus constraints
1479 and best-practice, which are somewhat different than the ones for a in-
1480 process JavaScript API.

1481 New vs Old Specification

- 1482 1. The [Vehicle Data Specification](#)⁵⁹ data model uses attributes (data) and
1483 interface objects, where VISS uses the [Vehicle Signal Specification](#)⁶⁰ data
1484 model which is based on a signal tree structure containing different entities
1485 types (branches, rbranches, signals, attributes, and elements).

⁵⁴<https://gitlab.apertis.org/pkg/rhosydd>

⁵⁵<https://www.w3.org/TR/vehicle-information-service/>

⁵⁶https://github.com/COVESA/vehicle_signal_specification

⁵⁷<https://www.w3.org/TR/vehicle-information-service/>

⁵⁸https://github.com/COVESA/vehicle_signal_specification

⁵⁹http://www.w3.org/2014/automotive/data_spec.html

⁶⁰https://github.com/COVESA/vehicle_signal_specification

- 1486 2. The [Vehicle Information Service Specification](#)⁶¹ API objects are defined as
1487 JSON objects that will be passed between the client and the VIS Server,
1488 where Rhosydd is currently based on accessing attributes values using
1489 interface objects.
- 1490 3. VISS defines a set of **Request Objects** and **Response Objects** (de-
1491 fined as JSON schemas), where the client must pass request messages to
1492 the server and they should be any of the defined request objects, in the
1493 same way, the message returned by the server must be one of the defined
1494 response objects.
- 1495 4. The request and response parameters contain a number of attributes,
1496 among them the Action attribute which specify the type of action re-
1497 quested by the client or delivered by the server.
- 1498 5. VISS lists well defined actions for client requests: authorize, getMetadata,
1499 get, set, subscribe, subscription, unsubscribe, unsubscribeAll.
- 1500 6. The [Vehicle Signal Specification](#)⁶² introduces the concept of **signals**. They
1501 are just named entities with a producer (or publisher) that can change its
1502 value over time and have a type and optionally a unit type defined.
- 1503 7. The [Vehicle Signal Specification](#)⁶³ data model introduces a signal specifica-
1504 tion format. This specification is a YAML list in a single file called **vspec**
1505 file. This file can also be generated in other formats (JSON, FrancaIDL),
1506 and basically defines the signal and data structure tree.
- 1507 8. The Vehicle Signal Specification introduces the concept of signal ID
1508 databases. These are generated from the vspec files, and they basically
1509 map signal names to ID's that can be used for easy indexing of signals
1510 without the need of providing the entire qualified signal name.

1511 Rhosydd New Changes

- 1512 • The [Vehicle Information Service Specification](#)⁶⁴ API defines the Request
1513 and Response Objects using a JSON schema format. The Rhosydd API
1514 (both the application-facing and backend-facing ones) has been updated
1515 to provide a similar API based on idiomatic DBus methods and types.
- 1516 • Maps the different VISS Server actions to handle client requests to their
1517 respective DBus methods in Rhosydd.
- 1518 • The internal Rhosydd data model has been updated to support all the
1519 element types defined in the [Vehicle Signal Specification](#)⁶⁵.

⁶¹<https://www.w3.org/TR/vehicle-information-service/>

⁶²https://github.com/COVESA/vehicle_signal_specification

⁶³https://github.com/COVESA/vehicle_signal_specification

⁶⁴<https://www.w3.org/TR/vehicle-information-service/>

⁶⁵https://github.com/COVESA/vehicle_signal_specification

- 1520 • It might also be required to add support to process signal ID databases
1521 in order for Rhosydd to recognize signals specified by the Vehicle Signal
1522 Specification.

1523 Advantages

- 1524 • The new VISS spec is based on a WebSocket API, and it resembles more
1525 closely the inter-process mechanism based on D-Bus in Rhosydd rather
1526 than the previous JavaScript in-process mechanism defined by the previous
1527 specification.

1528 Conclusion

1529 The main effort will be about updating the internal Rhosydd data model to
1530 reflect the changes introduced in the [Vehicle Signal Specification](#)⁶⁶ data model,
1531 with the extended types and metadata.

1532 The D-Bus APIs, both on the application and backend sides, will need to be
1533 updated to map to the new data model. From a high-level point of view the
1534 old and new APIs are relatively similar, but a non-trivial amount of changes is
1535 expected to map the new concepts and to align to the new terminology.

1536 The [Rhosydd](#)⁶⁷ client APIs for applications (librhosydd) and backends (libcroe-
1537 sor) will need to be updated to reflect the changes in the underlying D-Bus
1538 APIs.

1539 Appendix: W3C API

1540 For the purposes of completeness, the [Vehicle Information Service Specifica-
1541 tion](#)⁶⁸ is reproduced below. This is the version from the Final Business Group
1542 Report 26 June 2018, and does not include the [Vehicle Signal Specification](#)⁶⁹ for
1543 brevity. The API is described as [WebIDL](#)⁷⁰, and partial interfaces have been
1544 merged.

```
1545 [Constructor,  
1546   Constructor(VISClientOptions options)]  
1547 interface VISClient {  
1548   readonly attribute DOMString? host;  
1549   readonly attribute DOMString? protocol;  
1550   readonly attribute unsigned short? port;  
1551  
1552   [NewObject] Promise< void> connect();  
1553   [NewObject] Promise< unsigned long> authorize(object tokens);
```

⁶⁶https://github.com/COVESA/vehicle_signal_specification

⁶⁷<https://gitlab.apertis.org/pkg/rhosydd>

⁶⁸<https://www.w3.org/TR/vehicle-information-service/>

⁶⁹https://github.com/COVESA/vehicle_signal_specification

⁷⁰<http://www.w3.org/TR/WebIDL/>

```

1554     [NewObject] Promise< Metadata> getMetadata(DOMString path);
1555     [NewObject] Promise< VISValue> get(DOMString path);
1556     [NewObject] Promise< void> set(DOMString path, any value);
1557     VISSubscription subscribe(DOMString path, SubscriptionCallback subscriptionCallback, ErrorCallback errorCallback);
1558     [NewObject] Promise< void> unsubscribe(VISSubscription subscription);
1559     [NewObject] Promise< void> unsubscribeAll();
1560     [NewObject] Promise< void> disconnect();
1561 };
1562
1563 dictionary VISClientOptions {
1564     DOMString? host;
1565     DOMString? protocol;
1566     unsigned short? port;
1567 };
1568
1569 dictionary VISValue {
1570     any value;
1571     DOMTimeStamp timestamp;
1572 };
1573
1574 dictionary VISError {
1575     unsigned short number;
1576     DOMString? reason;
1577     DOMString? message;
1578     DOMTimeStamp timestamp;
1579 };
1580
1581 enum Availability {
1582     "available",
1583     "not_supported",
1584     "not_supported_yet",
1585     "not_supported_security_policy",
1586     "not_supported_business_policy",
1587     "not_supported_other"
1588 };

```