



Sensors and actuators

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

46	Triggering bundle activation	11
47	Bulk recording of sensor data	12
48	Sensor security	12
49	Actuator security	12
50	App store knowledge of device requirements	12
51	Accessing devices on multiple vehicles	12
52	Third-party accessories	13
53	SDK hardware support	13
54	Background on intra-vehicle networks	13
55	Existing sensor systems	13
56	W3C Vehicle Information Service Specification (VISS)	14
57	GENIVI Web API Vehicle	14
58	Apple HomeKit	14
59	Apple External Accessory API	15
60	iOS CarPlay	16
61	Android Auto	16
62	MirrorLink	16
63	Android Sensor API	17
64	Automotive Message Broker	17
65	AllJoyn	18
66	Approach	18
67	Overall architecture	19
68	Vehicle device daemon	19
69	Hardware and app APIs	20
70	Hardware API compliance testing	24
71	SDK API compliance testing and simulation	25
72	SDK hardware	25
73	Trip logging of sensor data	26
74	Properties vs devices	26
75	Property naming	27
76	High bandwidth or low latency sensors	27
77	Timestamps and uncertainty bounds	28
78	Registering triggers and actions	28
79	Bulk recording of sensor data	29
80	Security	29
81	Suggested roadmap	36
82	Requirements	37
83	Open questions	38
84	Summary of recommendations	39
85	Sensors and Actuators API	40
86	Rhosydd API Current State	40
87	Considerations to align Rhosydd to the new VISS API	40
88	New vs Old Specification	41
89	Rhosydd New Changes	41
90	Advantages	42

91	Conclusion	42
92	Appendix: W3C API	42

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, motorbike,
 108 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 de-
 115 vices which are supported by the network have messages or bandwidth allocated
 116 for them at the time of manufacture. No devices which are not known at the
 117 time 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_](http://www.w3.org/2014/automotive/vehicle_spec.html#h2_abstract)
156 [abstract.](http://www.w3.org/2014/automotive/vehicle_spec.html#h2_abstract))

157 **Trailer**

158 The driver attaches a trailer to their vehicle and it contains tyre pressure sensors.
159 These should be available to the virtual mechanic bundle.

160 **Petrol station finder**

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

164 **Sightseeing application bundle**

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

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

172 **Basic model vehicle**

173 If a vehicle does not have a compass sensor, the sightseeing bundle cannot
174 function at all, and the Apertis store should not allow the user to install it on
175 their vehicle.

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

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

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

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

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

192 **Privacy concern:** This should be implemented by granting access to overall
193 ‘volume level’ information for different zones in the vehicle; but *not* by grant-
194 ing access to the actual audio input data, which would allow the bundle to
195 record conversations. The overall volume level information should be sufficiently
196 smoothed or high-latency that a malicious application cannot infer audio infor-
197 mation from it.

198 **Night mode**

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

204 **Weather feedback or traffic jam feedback**

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

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

214 **Insurance bundle**

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

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

227 **Driving setup bundle**

228 An application bundle may want to control the driving setup — the position of
229 the steering wheel, its rake, the position of the wing mirrors, the seat position

230 and shape, whether the vehicle is in sport mode, etc. If a guest driver starts using
231 the vehicle, they could import their settings from the same bundle on their own
232 vehicle, and the bundle would automatically adjust the physical driving setup
233 in the vehicle to match the user's preferences. The bundle may want to restrict
234 these changes to only happen while the vehicle is parked.

235 **Odour detection**

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

242 **Air conditioning control**

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

248 **Automatic window feedback**

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

253 **Agricultural vehicle**

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

260 **Roof box**

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

266 **Truck installations**

267 Trucks are sold as a basis ‘vanilla’ truck with a special installation on top,
268 which is customised for the truck’s intended use. For example, a rubbish truck,
269 tipping truck or police truck. The installation is provided by a third party
270 who has a relationship with the basis truck manufacturer. Each installation
271 has specific sensors and actuators, which are to be controlled by an application
272 bundle provided by the third party or by the manufacturer.

273 **Compromised application bundle**

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

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

283 **Ethernet intra-vehicle network**

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

289 **Development against the SDK**

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

293 **Non-use-cases**

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

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

299 This is not a use case for the Sensors and Actuators API; it should be handled
300 by direct Bluetooth communication between the health monitor bundle and the
301 watch. If the Sensors and Actuators API were to support third-party devices

302 (as opposed to ones specified and installed by the vehicle manufacturer or sup-
303 pliers), having full support for all available devices would become a lot harder.
304 Additionally, devices would then appear and disappear while the vehicle was
305 running (for example, if the user turned off their watch's Bluetooth connection
306 while driving); this is not possible with fixed in-vehicle sensors, and would
307 complicate the sensor enumeration API.

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

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

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

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

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

333 **Buddied and vehicle fleet communications**

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

341 **Requirements**

342 **Enumeration of devices**

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

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

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

353 **Enumeration of vehicles**

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

357 It is expected that the set of vehicles may change dynamically while the vehicles
358 are running.

359 **Retrieving data from sensors**

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

367 **Sending data to actuators**

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

374 **Network independence**

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

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

378 **Bounded latency of processing sensor data**

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

388 **Extensibility for OEMs**

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

397 **Third-party backends**

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

405 **Third-party backend validation**

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

411 **Notifications of changes to sensor data**

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

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

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

420 **Uncertainty bounds**

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

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

428 **Failure feedback**

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

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

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

438 **Timestamping**

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

444 **Triggering bundle activation**

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

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

452 **Bulk recording of sensor data**

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

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

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

464 **Sensor security**

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

472 **Actuator security**

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

478 **App store knowledge of device requirements**

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

483 **Accessing devices on multiple vehicles**

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

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

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

491 **Third-party accessories**

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

495 **SDK hardware support**

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

500 This backend must not be available in target images.

501 **Background on intra-vehicle networks**

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

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

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

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

516 **Existing sensor systems**

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

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

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

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

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

536 **GENIVI Web API Vehicle**

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

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

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

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

553 **Apple HomeKit**

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

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

²https://github.com/GENIVI/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/>

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

560 At its core, HomeKit allows enumeration of devices (‘accessories’) in a home.
561 A large part of its API is dedicated to grouping these into homes, rooms, ser-
562 vice groups and zones so that collections of accessories can be interacted with
563 simultaneously.

564 Each accessory implements one or more ‘services’ which are defined interfaces
565 for specific functionality, such as a light switch interface, or a thermostat inter-
566 face. Each service can expose one or more ‘characteristics’ which are readable
567 or writeable properties of that interface, such as whether a light is on, the cur-
568 rent temperature measured by a thermostat, or the target temperature for the
569 thermostat.

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

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

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

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

582 **Apple External Accessory API**

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

586 In order to use the External Accessory API, an app must list the accessory
587 protocols it supports in its app manifest. Each accessory supports one or more
588 protocols, defined by the manufacturer, which are interfaces for aspects of the
589 device’s functionality. They are equivalent to the ‘services’ in the HomeKit API.
590 The code to implement these protocols is provided by the manufacturer, and
591 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>

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

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

597 **iOS CarPlay**

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

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

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

606 **Android Auto**

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

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

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

614 **MirrorLink**

615 [MirrorLink](#)¹⁵ is a proprietary system for integrating phones with the IVI display
616 — it is similar to iOS CarPlay and Android Auto, but produced by the [Car
617 Connectivity Consortium](#)¹⁶ rather than a device manufacturer like Apple or
618 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/>

619 The specifications for MirrorLink are proprietary and only available to registered
620 developers. In a brochure (now unavailable for download), it is stated that
621 support for allowing apps access to sensor data is planned for the future (as of
622 2014).

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

625 **Android Sensor API**

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

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

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

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

642 **Automotive Message Broker**

643 [Automotive Message Broker](#)²² is an Intel OTC project to broker information
644 from the vehicle networks to applications, exposing a [tweaked version](#)²³ of the
645 W3C Vehicle Information Access API (with a few types and naming conventions
646 tweaked) over D-Bus to apps, and interfacing with whatever underlying networks
647 are in use in the vehicle. In short, it has the same goals as the Apertis Sensors
648 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>

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

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

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

660 **AllJoyn**

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

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

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

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

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

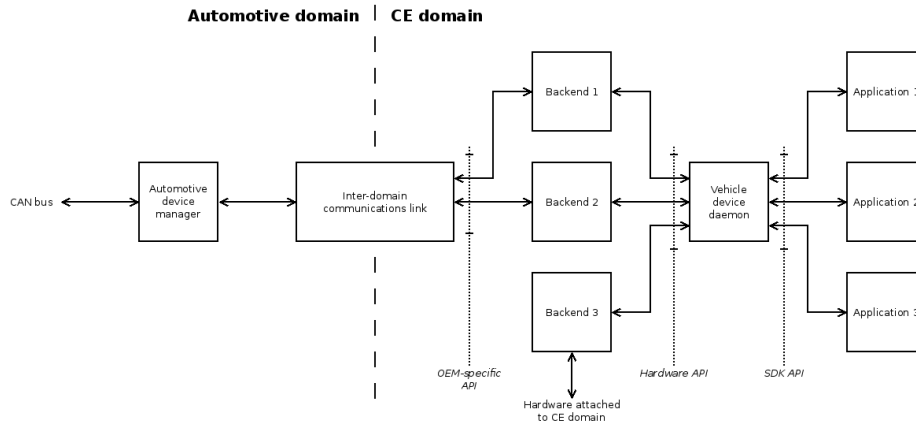
682 **Approach**

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

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

²⁵<https://openconnectivity.org/>

686 **Overall architecture**



687

688 **Vehicle device daemon**

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

697 The daemon needs to receive and process input within the latency bounds of
698 the sensors.

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

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

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

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

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

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

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

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

731 **Hardware and app APIs**

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

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

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

745 At least one backend service must be provided by the vehicle OEM, and it must
746 expose properties from the vehicle’s standard devices from the vehicle buses.
747 Other backend services may be provided by the vehicle OEM for other devices,
748 such as optional devices for premium vehicle models; or truck installations.

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>

749 Similarly, backend services may be provided by third parties for other devices,
750 such as after-market devices like roof boxes. Application bundles may provide
751 backend services as well, to expose hardware via application-specific protocols.
752 Consequently, backend services will likely be developed in isolation from each
753 other.

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

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

- 760 • Bulk enumeration of vehicles
- 761 • Bulk notification of changes to vehicle availability
- 762 • Bulk enumeration of properties of a vehicle, including readability and
763 writability
- 764 • Bulk notification of changes to property availability, readability or
765 writability
- 766 • Subscription to and unsubscription from property change notifications
- 767 • Bulk property change notifications for subscribed properties

768 The hardware API will be roughly a similar shape to the SDK API, and hence
769 a lot of complexity of the vehicle device daemon will be in the vehicle-specific
770 backends (both operate on properties — [Properties vs devices](#)).

771 As vehicle networks differ, the backend used in a given vehicle has to be de-
772 veloped by the OEM developing that vehicle. Apertis may be able to provide
773 some common utility functions to help in implementing backends, but cannot
774 abstract all the differences between vehicles. (See [Background on intra-vehicle
775 networks](#)).

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

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

³²<https://em.pages.apertis.org/apertis-website/concepts/inter-domain-communication/>

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

790 **Interactions between backend services**

791 In order to keep the security model for the system simple, backend services must
792 not be able to interact. Each device must be exposed by exactly one backend
793 service — two backend services cannot expose the same device; and neither can
794 they extend devices exposed by other backend services.

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

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

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

807 **Recommended hardware API design**

808 Below is a pseudo-code recommendation for the hardware API. It is not final,
809 but indicates the current best suggestion for the API. It has two parts — a
810 management API which is implemented by the vehicle device daemon; and a
811 property API which is implemented by each backend service and queried by the
812 vehicle device daemon.

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

814 **Management API**

815 Exposed on the well-known name `org.apertis.Rhosydd1` from the main daemon,
816 the `/org/apertis/Rhosydd1` object implements the standard `org.freedesktop.DBus.ObjectManager`³⁴
817 interface to let client discover and get notified about the registered vehicles.

818 Vehicles are mapped under `/org/apertis/Rhosydd1/{vehicle_id}` and implement
819 the `org.apertis.Rhosydd1.Vehicle` interface:

```
820 interface org.apertis.Rhosydd1.Vehicle {
```

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

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


```

821     readonly property s VehicleId;
822     method GetAttributes (
823         in s node_path,
824         out x current_time,
825         out a(s(vdx)a{sv}(uu)) attributes)
826     method GetAttributesMetadata (
827         in s node_path,
828         out x current_time,
829         out a(sa{sv}(uu)) attributes_metadata)
830     method SetAttributes (
831         in a{sv} attributes_value)
832     method UpdateSubscriptions (
833         in a(sa{sv}) subscriptions,
834         in a(sa{sv}) unsubscriptions)
835     signal AttributesChanged (
836         x current_time,
837         a(s(vdx)a{sv}(uu)) changed_attributes,
838         a(sa{sv}(uu)) invalidated_attributes)
839     signal AttributesMetadataChanged (
840         x current_time,
841         a(sa{sv}(uu)) changed_attributes_metadata)
842 }

```

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

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

- 850 • [Sunroof.Position](#)³⁵
- 851 • [Horn.IsActive](#)³⁶
- 852 • `Seat.FancySeatController.BackTemperature` (oem specific property)

853 Each attribute has three values associated:

- 854 • its value (of type v)
- 855 • its accuracy (as a standard deviation of type d, set to 0.0 for non-numeric
856 values)
- 857 • the timestamp when it was last updated (of type x)

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

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

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

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

- 861 • availability enum
 - 862 – AVAILABLE = 1
 - 863 – NOT_SUPPORTED = 0
 - 864 – NOT_SUPPORTED_YET = 2
 - 865 – NOT_SUPPORTED_SECURITY_POLICY = 3
 - 866 – NOT_SUPPORTED_BUSINESS_POLICY = 4
 - 867 – NOT_SUPPORTED_OTHER = 5
- 868 • access flags
 - 869 – NONE = 0
 - 870 – READABLE = (1 « 0)
 - 871 – WRITABLE = (1 « 1)

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

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

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

885 **Hardware API compliance testing**

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

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

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

- 901 • Success.
- 902 • Each failure code.
- 903 • Timeouts.
- 904 • Values which are out of range.

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

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

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

916 **SDK API compliance testing and simulation**

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

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

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

934 **SDK hardware**

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

939 This backend service must be separate from the emulator backend service (
940 [SDK API compliance testing and simulation](#)), in order to allow them to be used
941 independently.

942 **Trip logging of sensor data**

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

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

949 **Properties vs devices**

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

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

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

975 **Property naming**

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

- 986 • [Drivetrain.Transmission.Speed](#)³⁸
- 987 • [Drivetrain.Transmission.TravelledDistance](#)³⁹
- 988 • [DriveTrain.FuelSystem.TankCapacity](#)⁴⁰

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

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

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

999 **High bandwidth or low latency sensors**

1000 Sensors which provide high bandwidth outputs, or whose outputs must reach the
1001 bundle within certain latency bounds (as opposed to simply being aggregated
1002 by the vehicle device daemon within certain latency bounds), will be handled
1003 out of band. Instead of exposing the sensor data via the vehicle device daemon,
1004 the address of some out of band communications channel will be exposed. For
1005 video devices, this might be a V4L device node; for audio devices it might be a
1006 PulseAudio device identifier. Multiplexing access to the device is then delegated
1007 to the out of band mechanism.

1008 This considerably relaxes the performance requirements on the vehicle device

³⁷https://genivi.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/

1009 daemon, and allows the more specialist high bandwidth use cases to be handled
1010 by more specialised code designed for the purpose.

1011 **Timestamps and uncertainty bounds**

1012 The W3C Vehicle Signal Specification does not define uncertainty fields for
1013 any of its data types (for example, [VehicleSpeed](#)⁴² contains a single speed field
1014 measured in kilometres per hour). However, it allows the extensibility, so the
1015 data types exposed by the vehicle device daemon should all include an extension
1016 field specifying the uncertainty (accuracy) of the measurement, in appropriate
1017 units; and another specifying the timestamp when the measurement was taken,
1018 in monotonic time (in the [CLOCK_MONOTONIC](#)⁴³ sense).

1019 For example, the Apertis VehicleSpeed update looks like this:

```
1020 [ ('Drivetrain.Transmission.Speed',           -> property name  
1021     (110, 0.3, 38003116),                               -  
1022 > value field (speed, uncertainty, timestamp)  
1023   {'description': 'Latereal vehicle acceleration', -> metadata  
1024     'id': 54,  
1025     'type': 'Int32',  
1026     'unit': 'km/h'})  
1027 ]
```

1028 which represents a measurement of *speed* \pm *uncertainty* (110 \pm 0.3) kilometres
1029 per hour.

1030 **Registering triggers and actions**

1031 When subscribing to notifications for changes to a particular property using the
1032 [VehicleSignalInterface](#)⁴⁴ interface, a program is also subscribing to be woken up
1033 when that property changes, even if the program is suspended or otherwise not
1034 in the foreground.

1035 Once woken up, the program can process the updated property value, and poten-
1036 tially send a notification to the user. If the user interacts with this notification,
1037 the program may be brought to the foreground. The program must not be au-
1038 tomatically brought to the foreground without user interaction or it will steal
1039 the user's focus, which is distracting.

1040 See the draft compositor security design

1041 Alternatively, the program could process the updated property value in the
1042 background without notifying the user.

⁴²https://genivi.github.io/vehicle_signal_specification/rule_set/data_entry/sensor_actuator/

⁴³http://linux.die.net/man/3/clock_gettime

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

1043 The VehicleSignalInterface interface may be extended to support notifications
1044 only when a property value is in a given range; a degenerate case of this, where
1045 the upper and lower bounds of the range are equal, would support notifica-
1046 tions for property values crossing a threshold. This would most likely be imple-
1047 mented by adding optional min and max parameters to the VehicleSignalInter-
1048 face.subscribe() method.

1049 **Bulk recording of sensor data**

1050 This is a slightly niche use case for the moment, and can be handled by an
1051 application bundle running an agent process which is subscribed to the relevant
1052 properties and records them itself. This is less efficient than having the vehicle
1053 device daemon do it, as it means more processes waking up for changes in sensor
1054 data, but avoids questions of data formats to use and how and when to send bulk
1055 data between the vehicle device daemon and the application bundle's agent.

1056 If the implementation of this is moved into the vehicle device daemon, the
1057 lifecycle of recorded data must be considered: how space is allocated for the
1058 data's storage, when and how the application bundle is woken to process the
1059 data, and what happens when the allocated storage space is filled.

1060 **Security**

1061 The vehicle device daemon acts as a privilege boundary between all bundles
1062 accessing devices, between the bundles and the devices, and between each back-
1063 end service. Application bundles must request permissions to access sensor data
1064 in their manifest (see the Applications Design document), and must separately
1065 request permissions to interact with actuators. The split is because being able
1066 to control devices in the vehicle is more invasive than passively reading from
1067 sensors — it is safety critical. A sensible security policy may be to further split
1068 out the permissions in the manifest to require specific permissions for certain
1069 types of sensors, such as cabin audio sensors or parking cameras, which have
1070 the potential to be used for tracking the user. As adding more permissions
1071 has a very low cost, the recommendation is to err on the side of finer-grained
1072 permissions.

1073 The manifest should additionally separate lists of device properties which the
1074 bundle *requires* access to from device properties which it *may* access if they
1075 exist. This will allow the Apertis store to hide bundles which require devices
1076 not supported by the user's vehicle.

1077 From the permissions in the manifest, AppArmor and polkit rules restricting
1078 the program's access to the vehicle device daemon's API can be generated on
1079 installation of the bundle. See [Security domains](#) for rationale.

1080 When interacting with the vehicle device daemon, a program is securely identi-
1081 fied by its D-Bus connection credentials, which can be linked back to its man-
1082 ifest — the vehicle device daemon can therefore check which permissions the

1083 program’s bundle holds and accept or reject its access request as appropriate.
1084 Therefore, the vehicle device daemon acts as ‘the underlying operating system’ in
1085 controlling access, in the phrasing [used by](#)⁴⁵ the W3C specification. It enforces
1086 the security boundary between each bundle accessing devices, and between the
1087 intra- and inter-vehicle networks. The vehicle device daemon forms a separate
1088 security domain from any of the applications.

1089 Each backend service is a separate security domain, meaning that the vehicle
1090 device daemon is in a separate security domain from the intra-vehicle networks.

1091 The daemon may rate-limit API requests from each program in order to prevent
1092 one program monopolising the daemon’s process time and effectively causing a
1093 denial of service to other bundles by making API calls at a high rate. This
1094 could result from badly implemented programs which poll sensors rather than
1095 subscribing to change notifications from them, for example; as well as malicious
1096 bundles.

1097 Due to its complexity, low level in the operating system, and safety critical-
1098 ity, the vehicle device daemon requires careful implementation and auditing
1099 by an experienced developer with knowledge of secure software development at
1100 the operating system level and experience with relevant technologies (polkit,
1101 AppArmor, D-Bus).

1102 The threat model under consideration is that of a malicious or compromised
1103 bundle which can execute any of the D-Bus SDK APIs exposed by the daemon,
1104 with full manifest privileges for sensor access. A second threat model is that of
1105 a compromised backend service, which can execute any of the D-Bus hardware
1106 APIs exposed by the daemon.

1107 **Security domains**

1108 There are various security technologies available in Apertis for use in restricting
1109 access to sensors and actuators. See the Security Design for background on
1110 them; especially §9, Protecting the driver assistance system from attacks. These
1111 technologies can only be used on the boundaries between security domains. In
1112 this design, each application bundle is a single security domain (encompassing
1113 all programs in the bundle, including agents and helper programs); the vehicle
1114 device daemon is another domain; and each of the backend services are in a
1115 separate domain (including the vehicle networks they each use).

1116 **Application bundle and another application bundle or the rest of the** 1117 **system**

1118 Separation of the security domains of different application bundles from each
1119 other and from the rest of the system is covered in the Applications and Security
1120 designs.

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

1121 **Application bundle and vehicle device daemon**

1122 The boundary between an application bundle and the vehicle device daemon is
1123 the Sensors and Actuators SDK API, implemented by the daemon and exposed
1124 over D-Bus. The bundle's AppArmor profile will grant access to call any method
1125 on this interface if and only if the bundle requests access to one or more devices
1126 in its manifest. Note that AppArmor is not used to separate access to different
1127 sensors or actuators — it is not fine-grained enough, and is limited to allowing
1128 or denying access to the API as a whole.

1129 A separate set of [polkit](#)⁴⁶ rules for the bundle control which devices the bundle is
1130 allowed to access; these rules are generated from the bundle's manifest, looking
1131 at the specific devices listed. Given a set of polkit actions defined by the vehicle
1132 device daemon, these rules should permit those actions for the bundle.

1133 For example, the daemon could define the polkit actions:

- 1134 • org.apertis.vehicle_device_daemon.EnumerateVehicles: To list the avail-
1135 able vehicles or subscribe to notifications of changes in the list.
- 1136 • org.apertis.vehicle_device_daemon.EnumerateDevices: To list the avail-
1137 able devices on a given vehicle (passed as the vehicle variable on the action-
1138 tion) or subscribe to notifications of changes in the list.
- 1139 • org.apertis.vehicle_device_daemon.ReadProperty: To read a property,
1140 i.e. access a sensor, or subscribe to notifications of changes to the property
1141 value. The vehicle ID and property name are passed as the vehicle and
1142 property variables on the action.
- 1143 • org.apertis.vehicle_device_daemon.WriteProperty: To write a property,
1144 i.e. operate an actuator. The vehicle ID, property name and new value
1145 are passed as the vehicle, property and value variables on the action.

1146 The default rules for all of these actions must be polkit.Result.NO.

1147 If a bundle has access to any device, it is safe and necessary to grant it access to
1148 enumerate *all* vehicles and devices (the Enumerate* actions above) — otherwise
1149 the bundle cannot check for the presence of the devices it requires. Knowledge
1150 of which devices are connected to the vehicle should not be especially sensitive
1151 — it is expected that there will not be a sufficient variety of devices connected
1152 to a single vehicle to allow fingerprinting of the vehicle from the device list, for
1153 example.

1154 An application bundle, org.example.AccelerateMyMirror, which requests
1155 access to the vehicle.throttlePosition.value property (a sensor) and the vehi-
1156 cle.mirror.mirrorPan property (an actuator) would therefore have the following
1157 polkit rule generated in /etc/polkit-1/rules.d/20-org.example.AccelerateMyMirror.rules:

```
1158 polkit.addRule (function (action, subject) {  
1159     if (subject.credentials != 'org.example.AccelerateMyMirror') {
```

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

```

1160  /* This rule only applies to this bundle.
1161  * Defer to other rules to handle other bundles. */
1162  return polkit.Result.NOT_HANDLED;
1163  }
1164
1165  if (action.id == 'org.apertis.vehicle_device_daemon.EnumerateVehicles' ||
1166      action.id == 'org.apertis.vehicle_device_daemon.EnumerateDevices') {
1167      /* Always allow these. */
1168      return polkit.Result.YES;
1169  }
1170
1171  if (action.id == 'org.apertis.vehicle_device_daemon.ReadProperty' &&
1172      action.lookup ('property') == 'vehicle.throttlePosition.value') {
1173      /* Allow access to this specific property. */
1174      return polkit.Result.YES;
1175  }
1176
1177  if (action.id == 'org.apertis.vehicle_device_daemon.WriteProperty' &&
1178      action.lookup ('property') == 'vehicle.mirror.mirrorPan') {
1179      /* Allow access to this specific property,
1180       * with user authentication. */
1181      return polkit.Result.AUTH\_USER;
1182  }
1183
1184  /* Deny all other accesses. */
1185  return polkit.Result.NO;
1186  });

```

1187 In the rules, the subject is always the program in the bundle which is requesting
1188 access to the device.

1189 **Open question:** What is the exact security policy to implement regarding
1190 separation of sensors and actuators? For example, bundle access to sensors
1191 could always be permitted without prompting by returning `polkit.Result.YES`
1192 for all sensor accesses; but actuator accesses could always be prompted to the
1193 user by returning `polkit.Result.AUTH_SELF`. The choice here depends on the
1194 desired user experience.

1195 **Vehicle device daemon and a backend service**

1196 The boundary between the vehicle device daemon and one of its backend services
1197 is the Sensors and Actuators hardware API, implemented by the daemon and
1198 exposed over D-Bus. The backend service's AppArmor profile will grant access
1199 to call any method on this interface. Note that AppArmor is not used to grant
1200 or deny permissions to expose particular properties — it is not fine-grained
1201 enough, and is limited to allowing or denying access to the API as a whole.

1202 In order to limit the potential for a compromised backend service to escalate its
1203 compromise into providing malicious sensor data for any sensor on the system,
1204 each backend service must install a file which lists the Vehicle Data properties
1205 it might possibly ever provide to the vehicle device daemon. The vehicle device
1206 daemon must reject properties from a backend service which are not in this list.
1207 The list must not be modifiable by the backend service after installation (i.e. it
1208 must be read-only, readable by the vehicle device daemon).

1209 Furthermore, if a backend service is found to be exploitable after being deployed,
1210 it must be possible for the vehicle device daemon to disable it. This is expected
1211 to typically happen with backend services provided by application bundles, as
1212 opposed to those provided by OEMs or third parties (as these should go through
1213 stricter review, and disabling them would have a much larger impact). The
1214 vehicle device daemon must have a blacklist of backend services which it never
1215 loads. It must check the credentials of D-Bus messages from backend services
1216 against this blacklist.

1217 Using `GetConnectionCredentials`, which returns an unforgeable
1218 identifier for the peer: [http://dbus.freedesktop.org/doc/dbus-
1219 specification.html#bus-messages-get-connection-credentials](http://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-get-connection-credentials)

1220 In order to support one (vulnerable) version of a backend service being black-
1221 listed, but not the next (fixed) version, the blacklist must contain version num-
1222 bers, which should be compared against the installed version number of the
1223 backend service as listed in the system-wide application bundle manifest store.

1224 **Vehicle device daemon and the rest of the system**

1225 The vehicle device daemon itself must not be able to access any of the vehicle
1226 buses or any networks. It must be run as a unique user, which owns the daemon's
1227 binary, with its DAC permissions set such that other users (except root) cannot
1228 run it. It must not have access to any device files. See §9, Protecting the driver
1229 assistance system from attacks, of the Security design for more details.

1230 **Backend service and another backend service or the rest of the system**

1231

1232 In order to guarantee it is the only program which can access a particular vehicle
1233 bus or network, each backend service should run as a unique user. The service's
1234 binary must be owned by that user, with its DAC permissions set such that
1235 other users (except root) cannot run it. Any device files which it uses for access
1236 to the underlying vehicle networks must be owned by that user, with their DAC
1237 permissions set such that other users cannot access them, and udev rules in place
1238 to prevent access by other users. If the backend needs access to a (local) network
1239 interface to communicate with the vehicle network buses, that interface must
1240 be put in a separate network namespace, and the `CLONE_NEWNET` flag used
1241 when spawning the backend service to put it in that namespace. This prevents
1242 the service from accessing other network interfaces; and prevents other processes

1243 from accessing the buses. See §9, Protecting the driver assistance system from
1244 attacks, of the Security design for more details.

1245 **SDK emulator**

1246 Typically, it should not be possible for one program to have access to both
1247 the vehicle device daemon’s SDK API and its hardware API (this access is
1248 controlled by AppArmor). However, the SDK emulator is a special case which
1249 needs access to both — so either this must be possible as a special case, or the
1250 SDK emulator must be split into a backend service process and a UI process,
1251 which communicate via another D-Bus connection.

1252 **Apertis store validation**

1253 Application bundles which request permissions to access devices must undergo
1254 additional checks before being put on the Apertis store. This is especially im-
1255 portant for bundles which request access to actuators, as those bundles are then
1256 potentially safety critical.

1257 **Checks for access to sensors**

1258 Suggested checks for bundles requesting read access to sensors:

- 1259 • The bundle does not send privacy-sensitive data to services outside the
1260 user’s control (for example, servers not operated by the user; see the [User
1261 Data Manifesto](#)⁴⁷), either via network transmission, logging to local stor-
1262 age, or other means, without the user’s consent. Any data sent *with* the
1263 user’s consent must only be sent to services which follow the User Data
1264 Manifesto. For example (this list is not exhaustive):
 - 1265 – Tracking the vehicle’s movements.
 - 1266 – Monitoring the user’s conversations (audio recording).
- 1267 • The bundle does not have access to uniquely identifiable information, such
1268 as a vehicle identification number (VIN). Any exceptions to this would
1269 need stricter review.
- 1270 • The bundle clearly indicates when it is gathering privacy-sensitive data
1271 from sensors. For example, a ‘recording’ light displayed in the UI when
1272 listening using a microphone.
 - 1273 1.

1274 **Checks for access to actuators**

1275 Suggested checks for bundles requesting write access to actuators:

- 1276 • The bundle does not additionally have network access.

⁴⁷<https://userdatamanifesto.org/>

- 1277 • Actuators are only operated while the vehicle is not driving. Any excep-
1278 tions to this would need even stricter review.
 - 1279 • Manual code review of the entire bundle’s source code by a developer
1280 with security experience. The entire source code must be made available
1281 for review by the bundle developer, as it is all run in the same security
1282 domain. For example (this list is not exhaustive):
 - 1283 – Looking for ways the bundle could potentially be exploited by an
1284 attacker.
 - 1285 – Checking that the bundle cannot use the actuator inappropriately
1286 during normal operation if it encounters unexpected circumstances.
1287 (For example, checking that arithmetic bugs don’t exist which could
1288 cause an actuator to be operated at a greater magnitude than in-
1289 tended by the bundle developer.)
- 1290 **Open question:** The specific set of Apertis store validation checks for bundles
1291 which access devices is yet to be finalised.

1292 **Checks for backend services**

1293 Suggested checks for backend services for the vehicle device daemon, whether
1294 they are provided by an OEM, a third party or as part of an application bundle:

- 1295 • The backend service does not additionally have network access.
- 1296 • The backend service does not have write access to any of the file system
1297 except devices it needs, and the D-Bus socket.
- 1298 • The backend service cannot access any more device nodes than it needs
1299 to support its devices.
- 1300 • Manual code review of the entire bundle’s source code by a developer
1301 with security experience. The entire source code must be made available
1302 for review by the bundle developer, as it is all run in the same security
1303 domain. For example (this list is not exhaustive):
 - 1304 – Looking for ways the backend service could potentially be exploited
1305 by an attacker.
 - 1306 – Checking that the backend service cannot use any of its actuator in-
1307 appropriately during normal operation if it encounters unexpected
1308 circumstances. (For example, checking that arithmetic bugs don’t
1309 exist which could cause an actuator to be operated at a greater mag-
1310 nitude than intended by the developer.)
- 1311 • The backend service’s D-Bus service is only accessible by the vehicle device
1312 daemon (as enforced by AppArmor).

- 1313 • If other software is shipped in the same application bundle, it must be
1314 considered to be part of the same security domain as the backend service,
1315 and hence subject to the same validation checks.
- 1316 • The backend service must pass the automated compliance test ([Hardware](#)
1317 [API compliance testing](#)).
- 1318 • The backend service must not expose any properties which are not sup-
1319 ported by the version of the vehicle device daemon which it targets as its
1320 minimum dependency (see [Vehicle device daemon](#) for information about
1321 the extension process).

1322 **Suggested roadmap**

1323 Due to the large amount of work required to write a system like this from
1324 scratch, it is worth exploring whether it can be developed in stages.

1325 The most important parts to finalise early in development are the SDK and hard-
1326 ware APIs, as these need to be made available to bundle developers and OEMs
1327 to develop bundles and the backend services. There seems to be little scope for
1328 finalising these APIs in stages, either (for example by releasing property access
1329 APIs first, then adding vehicle and device enumeration), as that would result in
1330 early bundles which are incompatible with multi-vehicle configurations.

1331 Similarly, it does not seem to be possible to implement one of the APIs before
1332 the other. Due to the fragmented nature of access to vehicle networks, the
1333 backend needs to be written by the OEM, rather than relying on one written
1334 by Apertis for early versions of the system.

1335 Furthermore, the security implementation for the vehicle device daemon must
1336 be part of the initial release, as it is safety critical.

1337 One area where phased development is possible is in the set of properties itself
1338 — initial versions of the daemon and backends could implement a small, core
1339 set of the properties defined in the [VSS Ontology \(VSSo\)](#)⁴⁸, and future versions
1340 could expand that set of properties as time is available to implement them. As
1341 each property is a public API, it must be supported as part of the SDK one it
1342 has appeared in a released version of the daemon, so it is important to design
1343 the APIs correctly the first time.

1344 Similarly, the scope for backend services could be expanded over time. Initial
1345 releases of the system could allow only backend services written by vehicle OEMs
1346 to be used; with later releases allowing third-party backend services, then ones
1347 provided by installed application bundles.

1348 The emulator backend service ([SDK API compliance testing and simulation](#))
1349 and any SDK hardware backend services ([SDK hardware](#)) should be imple-
1350 mented early on in development, as they should be relatively simple, and hav-

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

1351 ing them allows application developers to start writing applications against the
1352 service.

1353 Requirements

- 1354 • **Enumeration of devices:** The availability of known properties of the vehicle
1355 can be checked through the [Availability interface](#)⁴⁹. The W3C approach
1356 considers properties, rather than devices, to be the enumerable items, but
1357 they are mostly equivalent (see [Properties vs devices](#)).
- 1358 • **Enumeration of vehicles:** The availability of objects implementing the
1359 W3C Vehicle interface on D-Bus is exposed using an interface like the
1360 D-Bus ObjectManager API.
- 1361 • **Retrieving data from sensors:** Properties can be retrieved through the
1362 [VehicleInterface interface](#)⁵⁰. For high bandwidth sensors, or those with
1363 latency requirements for the end-to-end connection between sensor and
1364 bundle, data is transferred out of band (see [High bandwidth or low latency
1365 sensors](#)).
- 1366 • **Sending data to actuators:** Properties can be set through the [VehicleSig-
1367 nalInterface](#)⁵¹ interface. As with getting properties, data for high band-
1368 width or low latency sensors is transferred out of band.
- 1369 • **Network independence:** The vehicle device daemon abstracts access to the
1370 underlying buses, so bundles are unaware of it.
- 1371 • **Bounded latency of processing sensor data:** The vehicle device daemon
1372 should have its scheduling configuration set so that it can provide latency
1373 guarantees for the underlying buses.
- 1374 • **Extensibility for OEMs:** Extensions are standardised through Apertis and
1375 released in the next version of the Sensors and Actuators API for use by
1376 the OEM.
- 1377 • **Third-party backends:** Backend services for the vehicle device daemon
1378 can be installed as part of application bundles (either built-in or store
1379 bundles).
- 1380 • **Third-party backend validation:** Backend services must be validated be-
1381 fore being installed as bundles (see [Checks for backend services](#)).
- 1382 • **Notifications of changes to sensor data:** Property changes are notified
1383 via a publish–subscribe interface on [VehicleSignalInterface](#)⁵². Notification
1384 thresholds are supported by optional parameters on that interface.

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

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

- 1385 • **Uncertainty bounds:** The W3C API is extended to include uncertainty
1386 bounds for measurements.
- 1387 • **Failure feedback:** Through its use of **Promises**⁵³, the API allows for failure
1388 to set a property.
- 1389 • **Timestamping:** The W3C API is extended to include timestamps for mea-
1390 surements.
- 1391 • **Triggering bundle activation:** Programs are woken by subscriptions to
1392 property changes (see **Registering triggers and actions**).
- 1393 • **Bulk recording of sensor data:** **Not currently implemented**, but may
1394 be implemented in future as a straightforward extension to the API. See
1395 **Bulk recording of sensor data**.
- 1396 • **Sensor security:** Access to the Sensors and Actuators API is controlled by
1397 an AppArmor profile generated from permissions in the manifest. Access
1398 to individual sensors is controlled by a polkit rule generated from the same
1399 permissions. See **Security**.
- 1400 • **Actuator security:** As with **Sensor security**; sensors and actuators are
1401 listed and controlled by the polkit profile separately.
- 1402 • **App-store knowledge of device requirements:** As devices required by an
1403 application bundle are listed in the bundle's manifest (see **Security**), the
1404 Apertis store knows whether the bundle is supported by the user's vehicle.
- 1405 • **Accessing devices on multiple vehicles:** Each vehicle is exposed as a sepa-
1406 rate D-Bus object, each implementing the W3C Vehicle interface.
- 1407 • **Third-party accessories:** Properties for third-party accessories must be
1408 standardised through Apertis and exposed as separate interfaces on the
1409 vehicle object on D-Bus.
- 1410 • **SDK hardware support:** SDK hardware should be supported through a
1411 separate development-only backend service written specifically for that
1412 hardware.

1413 Open questions

- 1414 1. **Hardware and app APIs:** The exact definition of the SDK API is yet to
1415 be finalised. It should include support for accessing multiple properties in
1416 a single IPC round trip, to reduce IPC overheads.
- 1417 2. **Interactions between backend services:** The exact means for aggregating
1418 each property in the Vehicle Data specification is yet to be determined.
- 1419 3. **Security domains:** What is the exact security policy to implement re-
1420 garding separation of sensors and actuators? For example, bundle access

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

1421 to sensors could always be permitted without prompting by returning
1422 polkit.Result.YES for all sensor accesses; but actuator accesses could al-
1423 ways be prompted to the user by returning polkit.Result.AUTH_SELF.
1424 The choice here depends on the desired user experience.

1425 4. **Apertis store validation:** The specific set of Apertis store validation checks
1426 for bundles which access devices is yet to be finalised.

1427 **Summary of recommendations**

1428 As discussed in the above sections, we recommend:

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

- 1457 sensitive) or actuators (safety critical). Or when application bundles re-
1458 quest permissions to provide a vehicle device daemon backend service
1459 (safety critical).
- 1460 • Modifying the Apertis software installer to generate AppArmor rules to
1461 allow D-Bus calls to the vehicle device daemon if device properties are
1462 listed in the application bundle manifest.
 - 1463 • Modifying the Apertis software installer to generate polkit rules to grant
1464 an application bundle access to specific devices listed in the application
1465 bundle manifest.
 - 1466 • Implementing and auditing strict DAC and MAC protection on the vehicle
1467 device daemon and each of its backend services, and identity checks on all
1468 calls between them.
 - 1469 • Defining a feedback and standardisation process for OEMs to request new
1470 properties or device types to be supported by the vehicle device daemon's
1471 API.

1472 Sensors and Actuators API

1473 This sections aims to compare the current status of the Vehicle device daemon
1474 for the sensors and actuators SDK API ([Rhosydd](#)⁵⁴) with the latest W3C spec-
1475 ifications: the [Vehicle Information Service Specification](#)⁵⁵ API and the [Vehicle](#)
1476 [Signal Specification](#)⁵⁶ data model.

1477 It will also explain the required changes to align Rhosydd to the new W3C
1478 specifications.

1479 Rhosydd API Current State

1480 The current Rhosydd API is stable and usable implementing the [Vehicle Infor-](#)
1481 [mation Service Specification](#)⁵⁷ and using the data model specified by the [Vehicle](#)
1482 [Signal Specification](#)⁵⁸.

1483 Considerations to align Rhosydd to the new VISS API

- 1484 1. The original Vehicle API and the Rhosydd API don't exactly match 1:1 as
1485 the latter has been adapted to follow the inter-process D-Bus constraints
1486 and best-practice, which are somewhat different than the ones for a in-
1487 process JavaScript API.

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

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

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

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

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

1488 New vs Old Specification

- 1489 1. The [Vehicle Data Specification](#)⁵⁹ data model uses attributes (data) and
1490 interface objects, where VISS uses the [Vehicle Signal Specification](#)⁶⁰ data
1491 model which is based on a signal tree structure containing different entities
1492 types (branches, rbranches, signals, attributes, and elements).
- 1493 2. The [Vehicle Information Service Specification](#)⁶¹ API objects are defined as
1494 JSON objects that will be passed between the client and the VIS Server,
1495 where Rhosydd is currently based on accessing attributes values using
1496 interface objects.
- 1497 3. VISS defines a set of **Request Objects** and **Response Objects** (de-
1498 fined as JSON schemas), where the client must pass request messages to
1499 the server and they should be any of the defined request objects, in the
1500 same way, the message returned by the server must be one of the defined
1501 response objects.
- 1502 4. The request and response parameters contain a number of attributes,
1503 among them the Action attribute which specify the type of action re-
1504 quested by the client or delivered by the server.
- 1505 5. VISS lists well defined actions for client requests: authorize, getMetadata,
1506 get, set, subscribe, subscription, unsubscribe, unsubscribeAll.
- 1507 6. The [Vehicle Signal Specification](#)⁶² introduces the concept of **signals**. They
1508 are just named entities with a producer (or publisher) that can change its
1509 value over time and have a type and optionally a unit type defined.
- 1510 7. The [Vehicle Signal Specification](#)⁶³ data model introduces a signal specifica-
1511 tion format. This specification is a YAML list in a single file called **vspec**
1512 file. This file can also be generated in other formats (JSON, FrancaIDL),
1513 and basically defines the signal and data structure tree.
- 1514 8. The Vehicle Signal Specification introduces the concept of signal ID
1515 databases. These are generated from the vspec files, and they basically
1516 map signal names to ID's that can be used for easy indexing of signals
1517 without the need of providing the entire qualified signal name.

1518 Rhosydd New Changes

- 1519 • The [Vehicle Information Service Specification](#)⁶⁴ API defines the Request
1520 and Response Objects using a JSON schema format. The Rhosydd API

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

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

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

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

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

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

1521 (both the application-facing and backend-facing ones) has been updated
1522 to provide a similar API based on idiomatic Dbus methods and types.

1523 • Maps the different VISS Server actions to handle client requests to their
1524 respective Dbus methods in Rhosydd.

1525 • The internal Rhosydd data model has been updated to support all the
1526 element types defined in the [Vehicle Signal Specification](#)⁶⁵.

1527 • It might also be required to add support to process signal ID databases
1528 in order for Rhosydd to recognize signals specified by the Vehicle Signal
1529 Specification.

1530 Advantages

1531 • The new VISS spec is based on a WebSocket API, and it resembles more
1532 closely the inter-process mechanism based on D-Bus in Rhosydd rather
1533 than the previous JavaScript in-process mechanism defined by the previous
1534 specification.

1535 Conclusion

1536 The main effort will be about updating the internal Rhosydd data model to
1537 reflect the changes introduced in the [Vehicle Signal Specification](#)⁶⁶ data model,
1538 with the extended types and metadata.

1539 The Dbus APIs, both on the application and backend sides, will need to be
1540 updated to map to the new data model. From a high-level point of view the
1541 old and new APIs are relatively similar, but a non-trivial amount of changes is
1542 expected to map the new concepts and to align to the new terminology.

1543 The [Rhosydd](#)⁶⁷ client APIs for applications (librhosydd) and backends (libcroe-
1544 sor) will need to be updated to reflect the changes in the underlying Dbus
1545 APIs.

1546 Appendix: W3C API

1547 For the purposes of completeness, the [Vehicle Information Service Specifica-
1548 tion](#)⁶⁸ is reproduced below. This is the version from the Final Business Group
1549 Report 26 June 2018, and does not include the [Vehicle Signal Specification](#)⁶⁹ for
1550 brevity. The API is described as [WebIDL](#)⁷⁰, and partial interfaces have been
1551 merged.

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

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

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

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

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

⁷⁰<http://www.w3.org/TR/WebIDL/>

```

1552 [Constructor,
1553   Constructor(VISClientOptions options)]
1554 interface VISClient {
1555     readonly attribute DOMString? host;
1556     readonly attribute DOMString? protocol;
1557     readonly attribute unsigned short? port;
1558
1559     [NewObject] Promise< void> connect();
1560     [NewObject] Promise< unsigned long> authorize(object tokens);
1561     [NewObject] Promise< Metadata> getMetadata(DOMString path);
1562     [NewObject] Promise< VISValue> get(DOMString path);
1563     [NewObject] Promise< void> set(DOMString path, any value);
1564     VISSubscription subscribe(DOMString path, SubscriptionCallback subscriptionCallback, ErrorCallback errorCallback);
1565     [NewObject] Promise< void> unsubscribe(VISSubscription subscription);
1566     [NewObject] Promise< void> unsubscribeAll();
1567     [NewObject] Promise< void> disconnect();
1568 };
1569
1570 dictionary VISClientOptions {
1571     DOMString? host;
1572     DOMString? protocol;
1573     unsigned short? port;
1574 };
1575
1576 dictionary VISValue {
1577     any value;
1578     DOMTimeStamp timestamp;
1579 };
1580
1581 dictionary VISError {
1582     unsigned short number;
1583     DOMString? reason;
1584     DOMString? message;
1585     DOMTimeStamp timestamp;
1586 };
1587
1588 enum Availability {
1589     "available",
1590     "not_supported",
1591     "not_supported_yet",
1592     "not_supported_security_policy",
1593     "not_supported_business_policy",
1594     "not_supported_other"
1595 };

```