



Debug and logging

Contents

2	Terminology and concepts	3
3	Application bundle	3
4	Component	4
5	Trusted dealer	4
6	Use cases	4
7	Debug deterministic application on SDK	4
8	Debug non-deterministic application on SDK	4
9	Debug application on target	4
10	Debug application in the context of the whole system	4
11	Extract logs from a device under test	5
12	Trusted dealer can extract logs from a device post-production	5
13	Third party cannot extract logs from a device post-production	5
14	Logging storage space is limited in post-production	5
15	Record and replay logs for input to an application	5
16	Record and replay logs for sensors to the whole system	5
17	Performance profiling	6
18	Denial of service attack on logging	6
19	Private application log file	6
20	Non-use-cases	6
21	Record and replay logs for entire system behaviour	6
22	Requirements	6
23	Code debugger installable on development and target machines	6
24	Code debugger can be used remotely	7
25	Code record and replay tool installable on development and target	
26	machines	7
27	Whole system logs are aggregated and timestamped	7
28	Whole system logs are tagged by process and priority	7
29	Whole system logs are limited by priority and rotated	8
30	Extract whole system logs from target device	8
31	Extract whole system logs from target device in post-production	8
32	Protect access to whole system logs on production devices	8
33	Code record and replay tool can handle multiple processes	8
34	Record and replay SDK sensor data	8
35	Profiling tools installable on development and target machines	9
36	Rate limiting of whole system logs	9
37	Applications can write their own log files	9
38	Disk usage for each application is limited	9
39	Existing debug and logging systems	9
40	Approach	9
41	GDB and gdbserver	10
42	Record and Replay (rr)	10
43	systemd journal	10
44	Application log files	11
45	Diagnostic log and trace	11

46	Extracting logs from a post-production system	11
47	D-Bus monitoring	12
48	Trip logging of SDK sensor data	12
49	Security	13
50	Disk usage and performance	14
51	Profiling tools	15
52	Suggested roadmap	15
53	Requirements	15
54	Open questions	16
55	Summary of recommendations	16

56 This documents several approaches to debugging components of an Apertis sys-
57 tem, either during development, or in the field. This includes debugging tools
58 for reproducing and analysing problems; and logging systems for gathering data
59 about problems and about system behaviour.

60 The major considerations with a debugging and logging system are:

- 61 • Reproducibility: Many of the hardest problems to diagnose are ones which
62 are hard to reproduce. A set of debugging tools should make it easy to re-
63 produce problems, and certainly should not make the problems disappear
64 when being debugged.
- 65 • Timing: An important part of ensuring that problems are reproducible is
66 ensuring that timing effects are reproducible, which means that a debug-
67 ging system must have a low (almost zero) overhead, in order to avoid
68 disturbing timing effects. Secondly to this, it must allow the developer
69 to see the order in which events occurred during the course of a problem.
- 70 • Context: As well as helping reproducibility of a problem, a debugging
71 system should reduce the need to reproduce the problem in the first place
72 —by capturing as much contextual information about it on the initial
73 attempt at debugging.
- 74 • Confidentiality: Any system which logs information about a running sys-
75 tem must ensure that the logged data remains confidential apart from to
76 developers who need it for debugging. This may mean that logging is not
77 enableable on production systems.

78 Terminology and concepts

79 Application bundle

80 An *application bundle* is a group of functionally related components (services,
81 data or programs) installed as a unit. This matches the sense with which ‘app’
82 is typically used on mobile platforms such as Android and iOS. (See the Appli-
83 cations design document for the full definition.)

84 **Component**

85 An application bundle or system service.

86 **Trusted dealer**

87 An authorised vehicle dealer, garage or other sale or repair location which has
88 a business relationship with the vehicle manufacturer.

89 **Use cases**

90 A variety of use cases for scenarios where a component needs debugging, or where
91 logging data are needed, are given below. Particularly important discussion
92 points are highlighted at the bottom of each use case.

93 Some of these cases may be already solved in the Apertis distribution in its
94 current state. However, they will all have an effect, to a greater or lesser extent,
95 on this design.

96 **Debug deterministic application on SDK**

97 An application developer needs to be able to debug their application when
98 running it on the SDK, diagnosing crashes and looking at log output for that
99 particular application.

100 **Debug non-deterministic application on SDK**

101 An application developer is working on an application whose behaviour appears
102 non-deterministic (for example, due to using a lot of threads, or depending on
103 sensitive timing). They manage to reproduce a particular bug only occasionally,
104 but need to debug it further.

105 **Debug application on target**

106 An application developer needs to be able to debug their application when run-
107 ning it on the target device (connected to an SDK machine during development),
108 diagnosing crashes and looking at log output for that particular application.

109 **Debug application in the context of the whole system**

110 An application developer has a problem with their application which is depen-
111 dent on the state of the whole (integrated) target system, rather than just on
112 internal state in their application. They need to be able to correlate system
113 state with their application's internal state.

114 **Extract logs from a device under test**

115 An Apertis tester has observed a failure in a development vehicle while doing
116 field testing on it. They need to be able to extract logs from the vehicle after
117 the event, and examine them offline to diagnose the failure.

118 **Trusted dealer can extract logs from a device post-production**

119 A vehicle owner has brought their vehicle into the garage with a failure in the
120 IVI system. The trusted dealer at the garage extracts logs from the vehicle and
121 passes them to the vehicle vendor for analysis, potentially leading to a fix for
122 the problem in a subsequent release of the CE domain operating system for that
123 vehicle.

124 **Third party cannot extract logs from a device post-production**

125 A vehicle owner likes to tinker with their vehicle, and would like to look at the
126 logs which their trusted dealer can look at, in order to get more information
127 about reverse engineering the IVI system in their vehicle.

128 They must not be able to access these logs.

129 **Logging storage space is limited in post-production**

130 On a production vehicle, the amount of storage space available for logging is
131 limited, so the system should log only the most important or recent and relevant
132 messages, and not write other messages to persistent storage.

133 **Record and replay logs for input to an application**

134 An application developer has found a problem in their application which depends
135 on external input to it, and subtle timing sequences of that input. The input
136 includes sensor input (from the SDK API, over D-Bus), and user interactions
137 with the interface using the touchscreen and on-screen keyboard. This makes it
138 a hard problem to reproduce. They want to add a regression test for it to their
139 application, and want to automate it because reproducing the problem manually
140 is too hard. This regression test needs to perfectly reproduce the problem each
141 time it is run.

142 The application has more than one process (it has one or more agent processes,
143 in addition to the main UI); all the processes communicate with each other at
144 runtime.

145 **Record and replay logs for sensors to the whole system**

146 An Apertis tester wants to test the whole system against a variety of road trips,
147 but it would be a waste of time to repeatedly drive a vehicle around a real road
148 system in order to do repeat test runs. They want a replayable log file of all
149 the sensor inputs from the vehicle, which can be replayed to the whole Apertis

150 system on a development machine, to allow repeated testing of how the system
151 responds to those inputs.

152 **Performance profiling**

153 An application is performing poorly on the target device, and the developer
154 wants to diagnose the problem so they can fix it.

155 **Denial of service attack on logging**

156 A misbehaving or malicious application is submitting log messages as fast as it
157 can. This should not adversely affect system performance, or cause other log
158 messages to be prematurely dropped.

159 **Private application log file**

160 An application is being ported from another platform to Apertis, and it already
161 has its own logging infrastructure, storing log messages in a private log file. The
162 developers wish to keep this infrastructure, rather than (or as well as) integrating
163 with the Apertis logging infrastructure.

164 **Non-use-cases**

165 **Record and replay logs for entire system behaviour**

166 While [this use case][Record and replay logs for sensors to the whole system] is
167 legitimate, it becomes harder to record the *entire* system behaviour (as opposed
168 to just the inputs from the sensor system), as that starts to be affected by
169 differences in the components which are being tested if those components are
170 changed to test new features. For example, if the entire system behaviour were
171 recorded and replayed, it might not be possible to run a debugger on the system
172 while replaying a log, as the debugger would impact the replay state too much.

173 **Requirements**

174 **Code debugger installable on development and target machines**

175 A code debugger must be available in Apertis, and installable on development
176 and target machines so that it can be used by Apertis and application develop-
177 ers.

178 The tool must allow interactive walking through the stack, printing expressions,
179 and other common C debugging functions.

180 See [Debug deterministic application on SDK](#).

181 **Code debugger can be used remotely**

182 The code debugger must be usable remotely in real time, most likely with a
183 server component running on the target device, and a client component on the
184 developer's machine.

185 See [Debug application on target](#).

186 **Code record and replay tool installable on development and target**
187 **machines**

188 A code record and replay tool must be available in Apertis, and installable
189 on development and target machines so that it can be used by Apertis and
190 application developers.

191 The tool must allow recording all inputs to an Application from the kernel, plus
192 any other system behaviour which would influence the application's behaviour.
193 Those logs must be stored as files, and replayable many times.

194 When replaying logs, the developer must be able to use a debugger to investigate
195 problems.

196 See:

- 197 • [Debug non-deterministic application on SDK](#)
- 198 • [Debug application in the context of the whole system](#)
- 199 • [Record and replay logs for input to an application](#)

200 **Whole system logs are aggregated and timestamped**

201 All log messages from all system components and services must be directed to
202 a central logging repository, which must timestamp them all in order (so that
203 all the timestamps are directly comparable).

204 See [Extract logs from a device under test](#), [Debug application in the context of](#)
205 [the whole system](#).

206 **Whole system logs are tagged by process and priority**

207 All log messages from all system components and services must be tagged with
208 the name of the process which generated them, and their priority (for example,
209 'debug' versus 'warning' versus 'error'). This metadata must be available to the
210 developer to allow them to filter logs for relevant messages.

211 See:

- 212 • [Debug deterministic application on SDK](#)
- 213 • [Debug application on target](#)

214 **Whole system logs are limited by priority and rotated**

215 On a production vehicle, the log messages which are written to persistent storage
216 must be limited to only the most recent logs (according to some age cutoff) and
217 the most important logs (according to some priority cutoff). These cutoffs must
218 be configurable at production time.

219 It may be possible to keep all other log messages in memory while the vehicle
220 is running, for example to allow them to be uploaded to an online diagnosis
221 service in case of a fault. They must not, however, be written to disk.

222 See [Logging storage space is limited in post-production.](#)

223 **Extract whole system logs from target device**

224 The aggregated system log on a development target device must be accessible
225 by the developer, who must be able to copy it to their development machine
226 for analysis. The log does not necessarily have to be extractable in real time,
227 though that would be helpful.

228 See [Extract logs from a device under test.](#)

229 **Extract whole system logs from target device in post-production**

230 The aggregated system log on a production target device must be extractable
231 by a trusted dealer so that It can be sent to an Apertis developer for analysis.
232 Extracting the log may require physical access to the vehicle.

233 See [Trusted dealer can extract logs from a device post-production.](#)

234 **Protect access to whole system logs on production devices**

235 The aggregated system log on a production device must only be extractable by
236 a trusted dealer or other authorised representative of the vehicle manufacturer.

237 See [Third-party cannot extract logs from a device post-production.](#)

238 **Code record and replay tool can handle multiple processes**

239 The code record and replay tool must be able to record and replay a single log
240 for multiple processes, such as an application and its agents. They must all see
241 the same timing information.

242 See [Record and replay logs for input to an application.](#)

243 **Record and replay SDK sensor data**

244 It must be possible to record all D-Bus traffic to and from the SDK sensors API
245 for a given time period (a ‘trip’), and later replay that log to the whole system
246 instead of using current sensor data.

247 See [Record and replay logs for sensors to the whole system](#).

248 **Profiling tools installable on development and target machines**

249 A variety of profiling tools must be available in Apertis, and installable on
250 development and target machines so that they can be used by Apertis and
251 application developers.

252 See [Performance profiling](#).

253 **Rate limiting of whole system logs**

254 To prevent denial of service attacks on the system log, rate limiting must be
255 applied to log message submissions from each application. If an application
256 submits log messages at too high a rate, the extras must be dropped.

257 See [Denial of service attack on logging](#).

258 **Applications can write their own log files**

259 Application developers may choose to ignore or supplement the Apertis SDK
260 logging infrastructure with their own system which writes to a log file in their
261 application's storage space. This must be permitted, although the SDK does
262 not have to provide convenience API for it.

263 See [Private application log file](#).

264 **Disk usage for each application is limited**

265 An application is logging to its own private log file, rather than the system
266 journal. The system must constrain the amount of disk space the application
267 can use, so that it cannot prevent other applications from working by consuming
268 all free disk space. If the application consumes too much disk space, the system
269 may delete its files or prevent it from working.

270 See [Private application log file](#).

271 **Existing debug and logging systems**

272 **Open question:** What existing debug and logging systems are relevant to do
273 background research on?

274 **Approach**

275 Based on the above research (section 6) and requirements (section 5), we rec-
276 ommend the following approach as an initial sketch of a debug and logging
277 system.

278 **GDB and gdbserver**

279 For real-time debugging of applications, both on a local SDK system and on
280 a remote target system, GDB should be used. For debugging remote systems,
281 gdbserver should be set up on the remote system and GDB used as a client to
282 control it.

283 They are both available in the development repository and Flatpak SDK, and
284 hence installable on development and target devices.

285 **Record and Replay (rr)**

286 For debugging of non-deterministic problems and problems which depend on
287 context or state outside of the application, Mozilla's Record and Replay (rr)
288 tool should be used. It works by recording all input and output to a process
289 (especially the input and output via kernel APIs), and allowing that log to be
290 replayed while re-running the application. This eliminates all sources of non-
291 determinism in the replay, ensuring that the conditions which triggered the
292 original problem can be reproduced every time.

293 Crucially, rr works with D-Bus: as all socket input and output for an application
294 is recorded, this includes all D-Bus traffic —this is reproduced faithfully in any
295 re-runs of the application. As many of the Apertis SDK APIs are provided via
296 D-Bus, this is a crucial feature.

297 In addition, rr can record a group of processes to a single log, and replay to the
298 same group later on. This can be used for debugging an application together
299 with its agents, for example.

300 Note, however, that rr is a *replay* tool and not an *interactive* debugger —a devel-
301 oper cannot replay a log recorded against one version of an application with a
302 newer version of the application (for example, with changes which the developer
303 hopes will fix the bug they're investigating). This is because it would change
304 the program's output behaviour and hence its effects on external processes.

305 For example, consider a bug where a program is writing a network packet to
306 the wrong socket out of two it has open. rr has recorded the response from the
307 socket the program was originally sending to (the wrong socket) —when a fixed
308 version of the program is run, the log file rr is using will not have a response
309 stored for the second (correct) socket.

310 This must be available in the development repository and Flatpak SDK, and
311 hence installable on development and target devices.

312 **systemd journal**

313 All log output from processes on the target system should be sent to the systemd
314 journal, allowing it to provide a single source of log data for the entire system,
315 with all log messages in a single ordering. This includes debug messages, errors,
316 warnings, and other log output. All messages should be sent with a priority

level, plus additional metadata if relevant. The journal automatically adds the sending process' name to log entries.

When developing on a local SDK system, the log should be queried using the `journalctl` command line tool.

Application log files

If an application developer chooses to log their application's messages to a private log file instead of, or as well as, to the `systemd` journal, this is permitted. The SDK may not provide convenience APIs for doing this, other than its APIs for file input and output. For example, it is up to the application developer to implement rate limiting, log file rotation and vacuuming.

Applications must not be able to consume all available disk space and prevent the system or other applications from working.

Applications may only write to their own log files if they have permission to write to persistent storage, which is one of the standard permissions in the application manifest.

Diagnostic log and trace

When testing a component on a target system, the developer should use diagnostic log and trace (DLT) from GENIVI —this is a client-server system where the DLT daemon runs on the target system and forwards `systemd` journal messages over the network to the developer's system, where they are presented in the DLT Viewer UI, which allows filtering, ordering, and other analysis to be performed on the logs.

However, DLT is only as useful as the log messages sent to it by the components on the system. Certain components may need to be modified to emit more log messages.

The DLT daemon exposes itself on the network and on the serial port with no authentication, so must not be installed by default on production systems.

Extracting logs from a post-production system

For extracting logs from a post-production system, a new *journal export service* must be written which provides and authenticates access to the `systemd` journal.

This service would essentially run the `journalctl -o export`¹ command to retrieve a full copy of the system's logs in a stable format suitable for sending to another system for review.

The service would need to listen on some external interface which a trusted dealer could connect to. This could, for example, be a network port; or it could

¹<http://www.freedesktop.org/wiki/Software/systemd/export/>

352 be a physical connector on the IVI system's main board. In any case, the service
353 must require authentication before exporting any logs.

354 **Open question:** What external interface can the journal export service listen
355 on?

356 The authentication mechanism chosen depends partially on the characteristics of
357 the interface the service listens on. It would most likely be a [challenge-response](#)
358 [protocol](#)² issued by the journal export service, where the trusted dealer proves
359 knowledge of a secret which has been issued by the vehicle manufacturer.

360 **Open question:** Should the logs be exported in an encrypted form, to keep
361 them confidential while being stored by a trusted dealer?

362 D-Bus monitoring

363 As many of the Aptaris SDK APIs are provided via D-Bus, an easy way to see
364 what they're doing is to log all D-Bus traffic on the system and session buses.
365 This can then be exposed by the DLT Viewer (or the local journalctl tool) and
366 analysed.

367 A new *D-Bus logging service* (similar to the dbus-monitor tool, but presented
368 as a systemd service which is enableable by developers, and only on development
369 images) should be written which logs all traffic for a specified D-Bus bus to the
370 systemd journal.

371 Note that this does not allow for log replay. For specific cases, this will be
372 handled using [Trip logging of SDK sensor data](#).

373 Trip logging of SDK sensor data

374 In order to record 'trip logs' of the sensor data sent to and from the SDK sensor
375 API and the entirety of the rest of the system, a *D-Bus record and replay tool*
376 should be written. When recording, this could monitor the D-Bus session bus
377 and record all traffic to and from the sensor API. When replaying, it would
378 replace the SDK sensor service on the bus, and impersonate all its APIs, re-
379 playing responses from the log. This program must be aware of the semantics
380 of D-Bus messages so, for example, it would not store the serial number of a
381 message reply, but would instead use the serial number corresponding to the
382 method call at the time of replay. Similarly, it must be aware of common D-Bus
383 interfaces such as org.freedesktop.DBus.Properties and know that the value of
384 a property remains unchanged unless a notification signal has been emitted for
385 it.

386 One implementation option would be to implement this based on the dbus-
387 monitor code: log all messages to or from the sensors API, and extract ones
388 with known semantics, such as org.freedesktop.DBus.Properties method calls
389 and signals. The replay code would maintain a queue of pairs of (expected

²https://en.wikipedia.org/wiki/Challenge%E2%80%93response_authentication

method call, reply), and for each incoming method call, would return and remove the first matching reply from the queue; or would return an error otherwise. For calls to known interfaces like `org.freedesktop.DBus.Properties`, the property state would be emulated with the correct semantics. Asynchronous events, such as signal emissions from the sensors API, would be emitted at the appropriate time relative to their surrounding events, rather than based on the absolute timestamp they were originally logged at. For example, if the log contained a signal emission after method call A and before method reply B, that signal would only be emitted in the replayed log after the program under test had made method call A.

An alternative implementation, which would be faster to implement but less generic and hence could not be repurposed for logging other SDK services in future, would be to use `python-dbusmock`³ to build a specific mock service for the sensors API. This service would have full knowledge of the semantics of all the D-Bus messages it sent and received —the full sensors SDK API, rather than just the standard D-Bus interfaces. The log file would be generated similarly to in the first implementation —by monitoring and interpreting the D-Bus traffic for the sensors API. The file would contain an initial set of values for the properties of all the sensors, followed by timestamped updates to each value as it changed during logging.

A third, most-specific, implementation option, is to use the emulator backend service for the vehicle device daemon (See the Sensors and Actuators design), and feed the recorded trip logs to it. This has the advantage of re-using the vehicle device daemon’s SDK API, without having to mock it up. The emulator backend service has to be written anyway, in order to implement the sensors and actuators emulator (see section 8.4 of the Sensors and Actuators design, version 0.3.0). This would be the fastest implementation option, and the least re-usable.

Example trip files To give application developers some baseline situations to test against, it would be helpful if Apertis or OEM variants of it shipped with several example trip logs, demonstrating some common or uncommon driving situations which applications must handle.

Open question: Should example trip files be produced by Apertis, or by OEMs so they are specific to vehicles?

Security

The security issues from logging are all concerned with confidentiality of system information, which may include sensitive data from a variety of processes.

This data must be kept confidential, both within the system (for example, applications must not have access to the logs of any process which is not in their

³<https://github.com/martinpitt/python-dbusmock>

429 trust domain), and from external attackers.

430 On production devices, especially, access to full system logs is a valuable goal
431 for an attacker, as it gives insight into how the system is configured and further
432 potential attack targets. For this reason, it may be worthwhile considering
433 whether to reduce or disable logging on production systems.

434 Conversely, log entries from production devices are very useful for debugging
435 unreproducible post-production problems. Therefore, the choice of logging
436 verbosity on production systems becomes a trade-off between the risk of confi-
437 dentiality breaches, and the practicality of being able to debug problems.

438 **Open question:** What level of logging should be enabled for production sys-
439 tems versus development systems?

440 Disk usage and performance

441 Storing log entries persistently consumes an unbounded amount of disk space.
442 A limit must be applied to the number or age of log entries which are stored
443 before being dropped. The systemd journal must have a disk space or age limit
444 applied; this can be done by editing `/etc/systemd/journald.conf` and adding the
445 following, for example:

```
446 SystemMaxUse=100M
```

447 To limit the priority level of messages which are stored to disk, the following
448 configuration option can be used; it is highly recommended to set it to ‘debug’
449 on development systems and ‘error’ for production systems.

450 The full range of options is documented in `man 5 journald.conf`

```
451 MaxLevelStore=error
```

452 Logging must not have a large runtime overhead —each call from a process to
453 the logging API must be fast. Furthermore, rate limiting must be applied to
454 prevent a misbehaving application from overfilling the system logs. This can
455 be achieved using the following configuration options for the systemd journal;
456 the following values limit each process to at most 1000 messages in a given 30
457 seconds:

```
458 RateLimitInterval=30s
```

```
459 RateLimitBurst=1000
```

460 As discussed in the Robustness design, the journal should additionally be config-
461 ured to leave an amount of free space smaller than the reserved blocks of the file
462 system containing the log files, so that log messages can continue to be written
463 in low disk space conditions, allowing easier diagnosis of the problem:

```
464 SystemKeepFree=5%
```

465 **Profiling tools**

466 A variety of profiling tools should be packaged for the Apertis development
467 repository and Flatpak SDK:

- 468 • perf
- 469 • valgrind
- 470 • google-perftools
- 471 • strace
- 472 • ltrace
- 473 • systemtap
- 474 • gprof

475 **Suggested roadmap**

476 GDB and DLT are already packaged, so no further work is needed there; as are
477 all the profiling tools.

478 rr is not yet packaged, but should be.

479 Integration of everything into the systemd journal, plus adding additional debug
480 messages to various system services to improve debuggability of those services.

481 The journal export service, D-Bus logging service and D-Bus record and replay
482 tools are all self-contained, so could be produced individually as later stages in
483 the implementation.

484 **Requirements**

- 485 • **Code debugger installable on development and target machines:** GDB is
486 the debugger.
- 487 • **Code debugger can be used remotely:** GDB can be used with gdbserver.
- 488 • **Code record and replay tool installable on development and target ma-**
489 **chines:** rr is the record and replay tool.
- 490 • **Whole system logs are aggregated and timestamped:** All system logs are
491 forwarded to the systemd journal. D-Bus messages are logged to the
492 journal via a new D-Bus logging service.
- 493 • **Whole system logs are tagged by process and priority:** Done by the sys-
494 temd journal by default.
- 495 • **Whole system logs are limited by priority and rotated:** Done with suitable
496 configuration of the systemd journal.
- 497 • **Extract whole system logs from target device:** DLT is used to extract logs
498 and transfer them to a developer machine in real time.

- **Extract whole system logs from target device in post-production** New journal export service exposing an authenticated interface for exporting systemd journal logs.
- **Protect access to whole system logs on production devices:** Journal export service requires authentication.
- **Code record and replay tool can handle multiple processes:** rr supports logging and replaying to multiple processes.
- **Record and replay SDK sensor data:** D-Bus record and replay tool will be used for this.
- **Profiling tools installable on development and target machines:** Various profiling tools will be packaged.
- **Rate limiting of whole system logs:** Done with suitable configuration of the systemd journal.
- **Applications can write their own log files:** Allowed for any application which is allowed to write files.
- **Disk usage for each application is limited:** Each application must have its storage usage restricted by the system.

Open questions

- What existing debug and logging systems are relevant to do background research on?
- What external interface can the journal export service listen on?
- Should the logs be exported in an encrypted form, to keep them confidential while being stored by a trusted dealer?
- Should example trip files be produced by Apertis, or by OEMs so they are specific to vehicles?
- What level of logging should be enabled for production systems versus development systems?

Summary of recommendations

As discussed in the above sections, we recommend:

- Packaging Mozilla's Record and Replay (rr) tool for the development repository.
- Ensure that all system components and services are logging exclusively to the systemd journal.
- Configure the systemd journal to handle log expiry, rotation and priority storage levels to avoid consuming unbounded disk space.

- 534 • Potentially add more debug log messages to various system services to
535 give more context when debugging applications.
- 536 • Write a journal export service for exporting the systemd journal with
537 authentication from a production system.
- 538 • Write a D-Bus logging service for logging all D-Bus traffic to the systemd
539 journal to give more context when debugging applications.
- 540 • Write a D-Bus record and replay tool for producing trip logs from the
541 SDK sensor API.
- 542 • Audit the confidentiality of the systemd journal and ensure it is only ac-
543 cessible to developers and the journal export service.
- 544 • Write documentation on how to use the Apertis SDK logging API, and
545 advice for application developers who want to use their own logging sys-
546 tems.