



Debug and logging

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

46	Diagnostic log and trace . . . . .	10
47	Extracting logs from a post-production system . . . . .	10
48	D-Bus monitoring . . . . .	11
49	Trip logging of SDK sensor data . . . . .	11
50	Security . . . . .	12
51	Disk usage and performance . . . . .	13
52	Profiling tools . . . . .	14
53	Suggested roadmap . . . . .	14
54	Requirements . . . . .	14
55	Open questions . . . . .	15
56	Summary of recommendations . . . . .	15

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

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

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

## 79 Terminology and concepts

### 80 Application bundle

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

85 **Component**

86 An application bundle or system service.

87 **Trusted dealer**

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

90 **Use cases**

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

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

97 **Debug deterministic application on SDK**

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

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

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

106 **Debug application on target**

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

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

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

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

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

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

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

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

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

129 They must not be able to access these logs.

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

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

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

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

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

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

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

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

### 153 **Performance profiling**

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

### 156 **Denial of service attack on logging**

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

### 160 **Private application log file**

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

## 165 **Non-use-cases**

### 166 **Record and replay logs for entire system behaviour**

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

## 174 **Requirements**

### 175 **Code debugger installable on development and target machines**

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

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

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

182 **Code debugger can be used remotely**

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

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

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

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

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

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

197 See:

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

201 **Application logs available in Eclipse when run on the SDK**

202 When developing an application in Eclipse, the logging calls the application uses  
203 must send their output to the Eclipse console (i.e. stdout or stderr) rather than  
204 (or as well as) the SDK system's journal. This allows the developer to easily  
205 read those messages.

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

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

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

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

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

214 All log messages from all system components and services must be tagged with  
215 the name of the process which generated them, and their priority (for example,

216 ‘debug’ versus ‘warning’ versus ‘error’). This metadata must be available to the  
217 developer to allow them to filter logs for relevant messages.

218 See:

- 219 • [Debug deterministic application on SDK](#)
- 220 • [Debug application on target](#)

### 221 **Whole system logs are limited by priority and rotated**

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

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

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

### 230 **Extract whole system logs from target device**

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

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

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

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

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

### 241 **Protect access to whole system logs on production devices**

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

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

### 245 **Code record and replay tool can handle multiple processes**

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



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

#### 250 **Record and replay SDK sensor data**

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

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

#### 255 **Profiling tools installable on development and target machines**

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

259 See [Performance profiling](#).

#### 260 **Rate limiting of whole system logs**

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

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

#### 265 **Applications can write their own log files**

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

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

#### 271 **Disk usage for each application is limited**

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

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

#### 278 **Existing debug and logging systems**

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

## 281 **Approach**

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

### 285 **GDB and gdbserver**

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

290 They must both be available in the development repository, and hence installable  
291 on development and target devices.

### 292 **Record and Replay (rr)**

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

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

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

307 Note, however, that rr is a *replay* tool and not an *interactive* debugger — a de-  
308 veloper cannot replay a log recorded against one version of an application with a  
309 newer version of the application (for example, with changes which the developer  
310 hopes will fix the bug they’re investigating). This is because it would change  
311 the program’s output behaviour and hence its effects on external processes.

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

317 This must be available in the development repository, and hence installable on  
318 development and target devices.

## 319 **systemd journal**

320 All log output from processes on the target system should be sent to the systemd  
321 journal, allowing it to provide a single source of log data for the entire system,  
322 with all log messages in a single ordering. This includes debug messages, errors,  
323 warnings, and other log output. All messages should be sent with a priority  
324 level, plus additional metadata if relevant. The journal automatically adds the  
325 sending process' name to log entries.

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

328 If a program is run manually from a console, or from within Eclipse, all log  
329 output must also be sent to stdout or stderr so that it appears on the console  
330 or the [Eclipse console](#)<sup>1</sup>.

## 331 **Application log files**

332 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.  
333 The SDK may not provide convenience APIs for doing this, other than its APIs  
334 for file input and output. For example, it is up to the application developer to  
335 implement rate limiting, log file rotation and vacuuming.  
336

337 Applications must not be able to consume all available disk space and prevent  
338 the system or other applications from working. The safety measures to prevent  
339 this are detailed in the Robustness design document, and primarily involve  
340 putting each application's storage area in a separate file system.

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

## 344 **Diagnostic log and trace**

345 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  
346 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  
347 the DLT Viewer UI, which allows filtering, ordering, and other analysis to be  
348 performed on the logs.  
349  
350

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

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

---

<sup>1</sup><https://git.gnome.org/browse/libgsystem/tree/src/gsystem-log.c#n128>

## 356 **Extracting logs from a post-production system**

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

359 This service would essentially run the `journalctl -o export`<sup>2</sup> command to retrieve  
360 a full copy of the system's logs in a stable format suitable for sending to another  
361 system for review.

362 The service would need to listen on some external interface which a trusted  
363 dealer could connect to. This could, for example, be a network port; or it could  
364 be a physical connector on the IVI system's main board. In any case, the service  
365 must require authentication before exporting any logs.

366 **Open question:** What external interface can the journal export service listen  
367 on?

368 The authentication mechanism chosen depends partially on the characteristics of  
369 the interface the service listens on. It would most likely be a [challenge-response](#)  
370 [protocol](#)<sup>3</sup> issued by the journal export service, where the trusted dealer proves  
371 knowledge of a secret which has been issued by the vehicle manufacturer.

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

## 374 **D-Bus monitoring**

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

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

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

## 385 **Trip logging of SDK sensor data**

386 In order to record 'trip logs' of the sensor data sent to and from the SDK  
387 sensor API and the entirety of the rest of the system, a *D-Bus record and*  
388 *replay tool* should be written. When recording, this could monitor the D-Bus  
389 session bus and record all traffic to and from the sensor API. When replaying, it  
390 would replace the SDK sensor service on the bus, and impersonate all its APIs,  
391 replaying responses from the log. This program must be aware of the semantics

<sup>2</sup><http://www.freedesktop.org/wiki/Software/systemd/export/>

<sup>3</sup>[https://en.wikipedia.org/wiki/Challenge%E2%80%93response\\_authentication](https://en.wikipedia.org/wiki/Challenge%E2%80%93response_authentication)

392 of D-Bus messages so, for example, it would not store the serial number of a  
393 message reply, but would instead use the serial number corresponding to the  
394 method call at the time of replay. Similarly, it must be aware of common D-Bus  
395 interfaces such as `org.freedesktop.DBus.Properties` and know that the value of  
396 a property remains unchanged unless a notification signal has been emitted for  
397 it.

398 One implementation option would be to implement this based on the `dbus-`  
399 `monitor` code: log all messages to or from the sensors API, and extract ones  
400 with known semantics, such as `org.freedesktop.DBus.Properties` method calls  
401 and signals. The replay code would maintain a queue of pairs of (expected  
402 method call, reply), and for each incoming method call, would return and remove  
403 the first matching reply from the queue; or would return an error otherwise.  
404 For calls to known interfaces like `org.freedesktop.DBus.Properties`, the property  
405 state would be emulated with the correct semantics. Asynchronous events, such  
406 as signal emissions from the sensors API, would be emitted at the appropriate  
407 time relative to their surrounding events, rather than based on the absolute  
408 timestamp they were originally logged at. For example, if the log contained  
409 a signal emission after method call A and before method reply B, that signal  
410 would only be emitted in the replayed log after the program under test had  
411 made method call A.

412 An alternative implementation, which would be faster to implement but less  
413 generic and hence could not be repurposed for logging other SDK services in  
414 future, would be to use [python-dbusmock](https://github.com/martinpitt/python-dbusmock)<sup>4</sup> to build a specific mock service  
415 for the sensors API. This service would have full knowledge of the semantics  
416 of all the D-Bus messages it sent and received — the full sensors SDK API,  
417 rather than just the standard D-Bus interfaces. The log file would be generated  
418 similarly to in the first implementation — by monitoring and interpreting the  
419 D-Bus traffic for the sensors API. The file would contain an initial set of values  
420 for the properties of all the sensors, followed by timestamped updates to each  
421 value as it changed during logging.

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

### 430 **Example trip files**

431 To give application developers some baseline situations to test against, it would  
432 be helpful if Apertis or OEM variants of it shipped with several example trip

---

<sup>4</sup><https://github.com/martinpitt/python-dbusmock>

433 logs, demonstrating some common or uncommon driving situations which appli-  
434 cations must handle.

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

### 437 **Security**

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

440 This data must be kept confidential, both within the system (for example, ap-  
441 plications must not have access to the logs of any process which is not in their  
442 trust domain), and from external attackers.

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

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

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

### 453 **Disk usage and performance**

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

```
459 SystemMaxUse=100M
```

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

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

```
464 MaxLevelStore=error
```

465 Logging must not have a large runtime overhead — each call from a process to  
466 the logging API must be fast. Furthermore, rate limiting must be applied to  
467 prevent a misbehaving application from overfilling the system logs. This can  
468 be achieved using the following configuration options for the systemd journal;

469 the following values limit each process to at most 1000 messages in a given 30  
470 seconds:

```
471 RateLimitInterval=30s
```

```
472 RateLimitBurst=1000
```

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

```
477 SystemKeepFree=5%
```

### 478 **Profiling tools**

479 A variety of profiling tools should be packaged for the Apertis development  
480 repository:

- 481 • perf
- 482 • valgrind
- 483 • google-perftools
- 484 • strace
- 485 • ltrace
- 486 • systemtap
- 487 • gprof
- 488 1.

### 489 **Suggested roadmap**

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

492 rr is not yet packaged, but should be.

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

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

### 498 **Requirements**

- 499 • **Code debugger installable on development and target machines:** GDB is  
500 the debugger.
- 501 • **Code debugger can be used remotely:** GDB can be used with gdbserver.

- 502 • **Code record and replay tool installable on development and target ma-**  
503 **chines:** rr is the record and replay tool.
- 504 • **Application logs available in Eclipse when run on the SDK:** Outputting  
505 log entries to stdout or stderr if running on a console.
- 506 • **Whole system logs are aggregated and timestamped:** All system logs are  
507 forwarded to the systemd journal. D-Bus messages are logged to the  
508 journal via a new D-Bus logging service.
- 509 • **Whole system logs are tagged by process and priority:** Done by the sys-  
510 temd journal by default.
- 511 • **Whole system logs are limited by priority and rotated:** Done with suitable  
512 configuration of the systemd journal.
- 513 • **Extract whole system logs from target device:** DLT is used to extract logs  
514 and transfer them to a developer machine in real time.
- 515 • **Extract whole system logs from target device in post-production** New  
516 journal export service exposing an authenticated interface for exporting  
517 systemd journal logs.
- 518 • **Protect access to whole system logs on production devices:** Journal export  
519 service requires authentication.
- 520 • **Code record and replay tool can handle multiple processes:** rr supports  
521 logging and replaying to multiple processes.
- 522 • **Record and replay SDK sensor data:** D-Bus record and replay tool will be  
523 used for this.
- 524 • **Profiling tools installable on development and target machines:** Various  
525 profiling tools will be packaged.
- 526 • **Rate limiting of whole system logs:** Done with suitable configuration of  
527 the systemd journal.
- 528 • **Applications can write their own log files:** Allowed for any application  
529 which is allowed to write files.
- 530 • **Disk usage for each application is limited:** Each application writes to its  
531 own file system as in the Robustness design.

## 532 Open questions

- 533 • What existing debug and logging systems are relevant to do background  
534 research on?
- 535 • What external interface can the journal export service listen on?
- 536 • Should the logs be exported in an encrypted form, to keep them confiden-  
537 tial while being stored by a trusted dealer?



- 538 • Should example trip files be produced by Apertis, or by OEMs so they are  
539 specific to vehicles?
- 540 • What level of logging should be enabled for production systems versus  
541 development systems?

## 542 **Summary of recommendations**

543 As discussed in the above sections, we recommend:

- 544 • Packaging Mozilla's Record and Replay (rr) tool for the development  
545 repository.
- 546 • Ensure that all system components and services are logging exclusively to  
547 the systemd journal.
- 548 • Configure the systemd journal to handle log expiry, rotation and priority  
549 storage levels to avoid consuming unbounded disk space.
- 550 • Potentially add more debug log messages to various system services to  
551 give more context when debugging applications.
- 552 • Write a journal export service for exporting the systemd journal with  
553 authentication from a production system.
- 554 • Write a D-Bus logging service for logging all D-Bus traffic to the systemd  
555 journal to give more context when debugging applications.
- 556 • Write a D-Bus record and replay tool for producing trip logs from the  
557 SDK sensor API.
- 558 • Audit the confidentiality of the systemd journal and ensure it is only  
559 accessible to developers and the journal export service.
- 560 • Write documentation on how to use the Apertis SDK logging API, and  
561 advice for application developers who want to use their own logging sys-  
562 tems.