



Contributions

1 Contents

2	Suitability of contributions	2
3	Upstream First Policy	2
4	Upstream Early, Upstream Often	2
5	Extending Apertis	3
6	Adding components to Apertis	3
7	Dedicated Project Areas	5
8	Extending existing components	6
9	Adding designs to Apertis	6
10	Concept Design Document Template	9
11	Other important bits	11
12	Sign-offs	11
13	Privileged processes	11
14	Getting commit rights	11
15	The role of maintainers	12

16 Contribution Template 13

17 This guide covers the expectations and processes for Apertis developers wish-
18 ing to make contributions to the Apertis project and the wider open source
19 ecosystem. These policies should be followed by all developers, including core
20 and third party contributors. A [checklist](#)¹ is provided in conjunction with these
21 policies to aid contributors.

22 Suitability of contributions

23 Like most open source projects, Apertis requires contributions are submitted via
24 a process (which in the case of Apertis is defined below) to ensure that Apertis
25 continues to meet it's design goals and remain suitable for it's community of
26 users. In addition to design and technical implementation details, the suitability
27 of contributions will be checked to meet requirements in areas such as [coding](#)
28 [conventions](#)² and [licensing](#)³.

29 Upstream First Policy

30 Apertis is a fully open source GNU/Linux distribution that carries a lot of com-
31 ponents for which it is not the upstream. The goal of [upstream first](#)⁴ is to
32 minimize the amount of deviation and fragmentation between Apertis compo-
33 nents and their upstreams.

¹https://jwd.pages.apertis.org/apertis-website/policies/contribution_checklist/

²https://jwd.pages.apertis.org/apertis-website/policies/coding_conventions/

³<https://jwd.pages.apertis.org/apertis-website/policies/license-expectations/>

⁴<https://jwd.pages.apertis.org/apertis-website/policies/upstreaming/>

34 Deviation tends to duplicate work and adds a burden on the Apertis developers
35 when it comes to testing and updating to newer versions of upstream compo-
36 nents. Also, as the success of Apertis relies on the success of open source in
37 general to accommodate new use cases, it is actively harmful for Apertis to not
38 do its part in moving the state of the art forward.

39 It is the intention of Apertis to utilize existing open source projects to provide
40 the functionality required, where suitable solutions are available, over the cre-
41 ation of home grown solutions that would fragment the GNU/Linux ecosystem
42 further.

43 This policy should be taken into consideration when submitting contributions
44 to Apertis.

45 **Upstream Early, Upstream Often**

46 One mantra that can be often heard in Open Source communitis is “upstream
47 early, upstream often”. The approach that this espouses is to breakdown large
48 changes into smaller chunks, attempting to upstream a minimal implementation
49 before implementing the full breath of planned features.

50 Each open source community tends to be comprised of many developers, which
51 share some overlap between their goals, but may have very different focuses. It
52 is likely that other developers contributing to the project may have ideas about
53 how the features that you are planning may be better implemented, for example
54 to enable a broader set of use cases to utilise the feature. Submitting an early
55 minimal implementation allows the general approach to be assessed, opinions
56 to be sought and a concensus reached regarding the implementation. As it is
57 likely that some changes will be required, a minimal implementation minimizes
58 the effort required to take feedback into account.

59 Taking this approach a step further, it can often be instructive to share your
60 intention to implement larger features before starting. Such a conversation
61 might be started by sending an email to the projects devel [mailing list](#)⁵ saying:

62 Hi,

63

64 I'm attempting to use <project> to <task> for my project.

65

66 I'm thinking about doing <brief technical overview> to enable this usecase.

67

68 I'm open to suggestions should there be a better way to solve this.

69

70 Thanks,

71

72 <developer>

⁵<https://lists.apertis.org/>

73 This enables other experienced developers the chance to suggest approaches that
74 may prove to be the most efficient, saving effort in implementation and later in
75 review, or may point to missed existing functionality that can be used to solve
76 a given need without needing substantial development effort.

77 **Extending Apertis**

78 **Adding components to Apertis**

79 Apertis welcomes requests for new components to be added to the distribution
80 and can act as a host for projects where required, however the open source focus
81 of Apertis should be kept in mind and any proposed contributions need to both
82 comply with Apertis policies and present a compelling argument for inclusion.

83 Additional components can be categorised into 3 main groups:

- 84 • Existing upstream component available in Debian stable (with suitable
85 version)
- 86 • Existing upstream component, not available in debian stable
- 87 • New component on gitlab.apertis.org

88 There is a maintenance effort associated with any components added to Apertis,
89 as any components added will need to be maintained within the Apertis ecosys-
90 tem. The effort required to maintain these different categories of components
91 are very different. Prepackaged Debian components require a lot less mainte-
92 nance effort than packaging other existing upstream components. Developing a
93 new component on gitlab.apertis.org requires both the development and pack-
94 aging/maintenance to be carried out within Apertis, significantly raising the
95 effort required.

96 When looking for ways to fulfill a requirement there are a number of factors
97 that will increase the probability of a solution being acceptable to Apertis.

- 98 • Component already included in Debian stable: As Apertis is based on
99 Debian and already has processes in place to pull updates from this source.
100 The cost of inclusion is dramatically lower than maintaining packages
101 drawn from other sources, as a lot of the required effort to maintain the
102 package is being carried out within the Debian ecosystem.
- 103 • Proven actively maintained codebase: Poorly maintained codebases
104 present a risk to Apertis, increasing the chance that serious bugs or
105 security holes will go unnoticed. Picking a solution that has an active user
106 base, a developer community making frequent updates and/or is a mature
107 codebase that has undergone significant “in the field” testing makes
108 the solution more attractive for inclusion in Apertis. It is understood
109 that, whilst extensive, the Debian repositories are not all encompassing,
110 if proposing an existing open source component that isn’t currently
111 provided by Debian, being able to show that it is actively maintained will
112 be important.

- 113 • Best solution: In general, there exists more open source solutions than
114 there exists problems. To be in with a good chance of having a compo-
115 nent included in Apertis it will be required to explain why the chosen
116 solution represents the best option for Apertis. What is “best” is often
117 nuanced and will be affected by a number of factors, including integra-
118 tion/overlap with existing components and the size/number of dependen-
119 cies it has (especially if they aren’t currently in Apertis). It may be that
120 whilst a number of existing solutions exist, none of them are a good fit for
121 Apertis. This may suggest a new component is the best solution, though
122 adapting/extending one of the existing solutions should also be considered.

123 The Apertis distribution is supported by it’s members. As previously men-
124 tioned, in order to ensure that Apertis remains viable and correctly focused, it
125 is important that any additions to the main [Apertis projects](#)⁶ are justified and
126 can be shown to fill a specific and real use case. Maintaining the packaging,
127 updating the codebases of which Apertis is comprised and performing testing
128 on supported platforms is a large part of the effort needed to provide Apertis.
129 As a result, it will be necessary to either be able to provide a commitment to
130 support any packages proposed for inclusion in the main Apertis projects or
131 gain such a commitment from an existing member.

132 The Apertis development team commit to maintaining the packages included in
133 the references images. Packages may be added to the main package repositories
134 but not form part of the reference images. Such packages will be maintained on
135 a best effort basis, that is as long as the effort remains reasonable the Apertis
136 team will attempt to keep the package in a buildable state, however runtime
137 testing will not be performed. Should the package fail to build or runtime issues
138 are reported and significant effort be required to modify the package the original
139 or subsequent users of the package may be approached to help resource fixing
140 the package. Ultimately the package may be removed if a solution can not be
141 found. Likewise, should a different common solution for Apertis be chosen at a
142 later date, the package may be deprecated and subsequently removed.

143 Proposals for inclusion of new components are expected to be made in the form
144 of a written proposal. Such a proposal should contain the following information:

- 145 • Description of the problem which is being addressed
146 • Why the functionality provided by the proposed component is useful to
147 Apertis and it’s audience
148 • A review of the possible solutions and any advantages and disadvantages
149 that have been identified with them
150 • Why the proposed solution is thought to present the best way forward,
151 noting the points made above where relevant
152 • Whether any resources are to be made available to help maintain the
153 component.

⁶https://jwd.pages.apertis.org/apertis-website/policies/package_maintenance/

154 **Dedicated Project Areas**

155 An alternative to adding packages to the main Apertis project is to apply to
156 have a dedicated project area, where code specific to a given project can be
157 stored. Such an area can be useful for providing components that are highly
158 specific to a given project and/or as a staging area for modifications to core
159 packages that might later get folded back into the main area, either by changes
160 being submitted to the relevant Apertis component or after changes have been
161 [upstreamed](#)⁷ to the components main project. A dedicated area will allow a
162 project group to iterate on key components more rapidly as the changes made
163 do not need to work across the various supported hardware platforms. It must
164 be noted that whilst a dedicated project area would allow some requirements
165 with regard to platform support to be ignored, packages in such areas would still
166 be required to comply with other Apertis rules such as [open source licensing](#)⁸.
167 It should be expected that the Apertis developers will take a very hands off
168 approach to the maintenance and testing of packages in such areas. If packages
169 in such areas require work, the project maintainers will be contacted. The
170 Apertis maintainers may at their discretion help with minor maintenance tasks
171 should a package be of interest to the Apertis project. Packages that become
172 unmaintained may be removed.

173 Requests for dedicated project areas are also expected to be made in a form of
174 a written proposal. Such a proposal should contain the following information:

- 175 • Description of the project requiring a dedicated project area
- 176 • Preferred name to be used to refer to the project
- 177 • Expected use of the dedicated area
- 178 • Expected lifetime of the project area
- 179 • Contact details of project maintainers

180 Such submissions should be made via the devel [mailing list](#)⁹.

181 The submission should be discussed on the mailing list and must be agreed with
182 the Apertis stakeholders.

183 **Extending existing components**

184 Apertis carries a number of packages that have been modified compared to their
185 upstream versions. It is fairly typical for distributions to need to make minor
186 modifications to upstream sources to tailor them to the distribution, Apertis is
187 not different in this regard.

188 Whilst Apertis does accept changes to existing components, it needs to be ac-
189 knowledged that this increases the effort required to maintain the package in
190 question. It may be requested that an attempt be made to upstream the changes,
191 in line with the [upstream first](#) policy, either to the packages upstream or Debian.

⁷<https://jwd.pages.apertis.org/apertis-website/policies/upstreaming/>

⁸<https://jwd.pages.apertis.org/apertis-website/policies/license-expectations/>

⁹<https://lists.apertis.org/>

192 More guidance is provided in the [upstreaming](#)¹⁰ documentation. If changes are
193 not generally of use or would have a negative impact on the broader Apertis
194 user base, changes may be required to be carried by the specific project within
195 a [dedicated project area](#).

196 **Adding designs to Apertis**

197 Another way to contribute to Apertis is with design documents. A design docu-
198 ment contains the description of all relevant aspects of a feature or of a require-
199 ment. The current design documents can be found in the [Concepts Designs](#)
200 [section](#)¹¹. These documents cover topics that have been researched but not
201 necessarily implemented. They should provide a good understanding of the im-
202 pact of the technology that forms the basis of the concept, what it is, how it
203 works, what are the threat models, the required infrastructure, how it would be
204 integrated with Apertis and anything else that is deemed relevant.

205 Such designs should be updated when implemented to explicitly cover the fin-
206 nal implementation and moved to a suitable section of the site, typically the
207 [Architecture](#)¹² or [Guides](#)¹³ section.

208 Project-wide impact is the metric used to decide if a contribution will be handled
209 as a component or as a design. If the impact of the contribution on the Apertis
210 project goes beyond the additional maintenance effort, it is likely to require a
211 design document before the component contribution.

212 As an example we will consider a proposal to provide tools and workflows for
213 process automation by including the [Robot Framework](#)¹⁴ in the Apertis Uni-
214 verse. The Robot Framework is a generic open source automation framework
215 that can be used for automation of tests and processes. Robot Framework is
216 released under [Apache License 2.0](#)¹⁵. However we do not expect to ship Robot
217 Framework components on Apertis target images.

218 The first important consideration is the state-of-the-art for addressing the goals
219 of the design. In our example the Robot Framework is preferred due it's matu-
220 rity, unique and simple to use descriptive language, and it's active development
221 community. However a strong argument in favor of the Robot Framework is it's
222 user base. Adding the Robot Framework to the Apertis Universe is expected to
223 bring Robot Framework users to Apertis.

224 The next important consideration are how the design is expected to work and
225 the potential impact on Apertis. The Robot framework has a layered archi-
226 tecture. The top layer is the simple, powerful, and extensible keyword-driven
227 descriptive language for testing and automation. This language resembles a

¹⁰<https://jwd.pages.apertis.org/apertis-website/policies/upstreaming/>

¹¹<https://jwd.pages.apertis.org/apertis-website/concepts/>

¹²<https://jwd.pages.apertis.org/apertis-website/architecture/>

¹³<https://jwd.pages.apertis.org/apertis-website/guides/>

¹⁴<https://robotframework.org/>

¹⁵<http://www.apache.org/licenses/LICENSE-2.0.html>

228 natural language, is quick to develop, is easy to reuse, and is easy to extend.
229 On the bottom layer of the architecture is the item to be tested, or the process
230 to be automated.

231 The middle layer is what makes the Robot Framework extensible: libraries.
232 A library, in Robot Framework terminology, extends the Robot Framework
233 language with new keywords, and provides the implementation for these new
234 keywords. Each Robot Framework library acts as glue between the high level
235 language and low level details of the item being tested, or of the environment
236 in which the item to be tested is present.

237 Adding the Robot Framework to the Apertis Universe has potential to impact:

- 238 1. Development workflow: Apertis encourages the use of continuous integra-
239 tion and the use of shared infrastructure resources instead of resources
240 that are private to specific developers.
- 241 2. Testing Apertis images: Apertis encourages the use of environments that
242 are as close as possible to production environments, meaning that ideally,
243 the Apertis images under test are not instrumented for testing, and are
244 only minimally modified.
- 245 3. Testing infrastructure: Apertis uses LAVA for deployment of operating sys-
246 tem and software in hardware, and for automated testing. The two main
247 constraints are LAVA being asynchronous and non-interactive. While both
248 developers and CI pipelines can submit jobs to LAVA, they cannot inter-
249 act with a job while it is running. The LAVA workflow is: submit a job,
250 wait for the job to be selected for execution, wait for the job to complete
251 execution, and download test results.

252 Addressing the benefits of the new design proposal is also important. As men-
253 tioned, adding tools and workflows for process automation with the Robot
254 Framework will extend the Apertis projects and we expect to attract more
255 users by doing so. Adding real-world use cases can illustrate the value with a
256 good level of details.

257 The proposal should also describe how to address the integration with Apertis
258 taking into account the constraints of the Apertis development workflow, of
259 testing Apertis images, and of the Apertis testing infrastructure.

260 The design proposal can also include a high level description of the estimated
261 work. For example, adding Robot Framework to Apertis will involve developing
262 and/or modifying Robot Framework libraries; and developing a run-time com-
263 patibility layer for LAVA to keep testing environments as close as possible to
264 production environments, and to adapt the execution of Robot Framework tests
265 to suit the LAVA constraints.

266 And finally it could contain a high level implementation plan. In our example,
267 one possible way to integrate Robot Framework is to adopt it in stages:

- 268 1. Add Robot Framework to the Apertis SDK to enable developers to use
269 the Robot Framework locally

- 270 2. Robot Framework Integration development: Adapt libraries and create
271 the run-time compatibility layer for LAVA
272 3. Deployment on the Apertis infrastructure

273 This section describes general topics, but it may not be complete for all designs.
274 Regarding the level of details the design document should be complete enough
275 to describe the design and surrounding problems to developers and project man-
276 agers, but it is not necessary to describe implementation details.

277 As a rule of thumb start with a lean design document and submit it for review
278 as early as possible. You can send a new design for review to the same process
279 used for a [component contribution](#)¹⁶.

280 **Concept Design Document Template**

281 The following template should be used as a guide when writing new concept
282 designs:

¹⁶https://jwd.pages.apertis.org/apertis-website/guides/development__process/

```
1  +++
2  title = "<document title>"
3  weight = 100
4  outputs = [ "html", "pdf-in",]
5  date = "20xx-xx-xx"
6  +++
7
8  # Introduction
9
10 # Terminology and concepts
11
12 # Use cases
13
14 # Non-use cases
15
16 # Requirements
17
18 # Existing systems
19
20 # Approach
21
22 # Evaluation Report
23
24 # Recommendation
25
26 ## Design recommendations
27
28 # Alternative designs
29
30 # Open questions
31
32 ## Unresolved design questions
33
34 ## Unresolved implementation questions
35
36 # Risks
37
38 # Summary
39
40 # Appendix
41
42 # References
```

283 Other important bits

284 Sign-offs

285 Like the git project and the Linux kernel, Apertis requires all contributions to
286 be signed off by someone who takes responsibility for the open source licensing
287 of the code being contributed. The aim of this is to create an auditable chain
288 of trust for the licensing of all code in the project.

289 Each commit which is pushed to git master **must** have a `Signed-off-by` line,
290 created by passing the `--signoff/-s` option to `git commit`. The line must give
291 the real name of the person taking responsibility for that commit, and indicates
292 that they have agreed to the [Developer Certificate of Origin](#)¹⁷. There may be
293 multiple `Signed-off-by` lines for a commit, for example, by the developer who
294 wrote the commit and by the maintainer who reviewed and pushed it:

```
295 Signed-off-by: Random J Developer <random@developer.example.org>  
296 Signed-off-by: Lucky K Maintainer <lucky@maintainer.example.org>
```

297 Apertis closely follows the Linux kernel process for sign-offs, which is described
298 in section 11 of the [kernel guide to submitting patches](#)¹⁸.

299 Privileged processes

300 Pushing commits to `gitlab.apertis.org` requires commit rights. Whilst commit
301 rights to most repositories are only granted to trusted contributors (see “[Getting](#)
302 [commit rights](#)” for how to get commit rights) the Apertis GitLab infrastructure
303 is open for registration, enabling anyone to sign up for an account, fork packages
304 into their personal space and submit merge requests (see the [development pro-](#)
305 [cess](#)¹⁹ for more details). All commits must have a `Signed-off-by` line assigning
306 responsibility for their open source licensing.

307 Some admin steps on the periphery of packaging and releasing new versions of
308 Apertis modules as Debian packages may require access to `build.collabora.co.uk`
309 (OBS). These are issued separately from commit rights, and are generally not
310 needed for the main development workflows.

311 Submitting automated test runs on `lava.collabora.co.uk` requires CI rights,
312 which are granted similarly to packaging rights. However, CI results may be
313 viewed read-only by anyone.

314 Getting commit rights

315 Commit rights (to allow direct pushes to git, and potentially access to the
316 package building system, `build.collabora.co.uk`) may be granted to trusted third

¹⁷<http://developercertificate.org/>

¹⁸<https://www.kernel.org/doc/Documentation/SubmittingPatches>

¹⁹https://jwd.pages.apertis.org/apertis-website/guides/development__process/

317 party contributors if they regularly contribute to Apertis, with high quality
318 contributions at the discretion of current Apertis maintainers.

319 Accounts on the Apertis GitLab instance can be available via [open registra-
320 tion](#)²⁰

321 By creating an account you signify that you accept the Apertis [Privacy Policy](#)²¹
322 and [Terms of Use](#)²²

323 For access to other Apertis infrastructure, please send an email to `account-
324 requests@apertis.org` including:

- 325 • Your full name
- 326 • The email address you prefer to be contacted through
- 327 • The nickname/account name you wish to be known by on the Apertis
328 GitLab

329 **The role of maintainers**

330 Most Open Source projects have one or more core contributors that take on a
331 managerial role for the project. This group may include the original author(s)
332 of the project and long-term trusted contributors, though in many projects with
333 a longer history, lead of the project may well have been taken on by another
334 knowledgeable contributor.

335 The basic role of a project maintainers is to:

- 336 • help set the direction for the project;
- 337 • ensure that the projects policies are followed and that the project continues
338 to work towards it's stated objectives;
- 339 • review and evaluate contributions for correctness and suitability;
- 340 • apply accepted contributions;
- 341 • resolve issues (such as bugs and security issues) that arise;
- 342 • and ensure the processes required to release new project artifacts are com-
343 pleted.

344 Larger projects may have many maintainers who specialise in parts of the work
345 that need to be carried out or who have deeper knowledge of specific parts of
346 a larger codebase. For example such maintainers may be in charge of applying
347 these roles to a single component within the Apertis distribution.

348 The Apertis maintainers are funded by the projects backers, with direction
349 agreed between the maintainers and backers to fullfill the needs of the backers
350 whilst driving the project towards it's stated objectives. Many of the maintainers
351 have a long history with the Apertis project or have come to the project with
352 lots of experience in the area in which they work (such as Debian packaging).

²⁰https://gitlab.apertis.org/users/sign_up

²¹https://jwd.pages.apertis.org/apertis-website/policies/privacy_policy/

²²https://jwd.pages.apertis.org/apertis-website/policies/terms_of_use/

353 The Apertis maintainers are responsible for ensuring that bug and security fixes
354 are applied to the various components of which Apertis is made and for migrat-
355 ing to newer releases of it's upstreams inline with the documented polices. The
356 maintainers then ensure that the source of these components is reliably built
357 into the binaries and images provided, covering the range of architectures and
358 platforms supported by the project.

359 In addition to tracking updates and fixes from the projects that Apertis uses, the
360 maintainers also review changes that are submitted to the project from contrib-
361 utors. The maintainers actively contribute to the project and submit changes
362 following the same processes that are expected from other contributors. All
363 such changes are reviewed to ensure that they meet the project goals, objectives
364 and policies as well as ensuring the are sound and do not contain any obvious
365 issues.

366 Whilst some contributors may remain active within the projects community
367 of users and developers for some time, this is a long way from guaranteed.
368 Maintainers must evaluate contributions to ensure that the changes that are
369 being proposed would continue to be maintainable in the absense of the original
370 contributor. As a result the maintainers may reject contributions that otherwise
371 appear to meet the policies if they feel that they would be impossible to maintain
372 or requiring changes to make the contribution more maintainable for the project.

373 The maintainer is usually taking on the responsibility on behalf of the project
374 to ensure that your changes and modifications continue to be provided by the
375 project, porting them to new versions of packages or ensuring that they remain
376 valid as the project inevitably changes to accomodate new goals or the ever
377 changing computing landscape. As a result accepting changes will transfer this
378 burden from you to the maintainers. You can continue to use the project with-
379 out needing to actively maintain the changes. As a result the onus is on the
380 contributor to persuade the project of the advantages of the changes, not for
381 the project to be beholden to accept contributions.

382 **Contribution Template**

383 This section contains a contribution template that illustrates the ideal first email
384 a developer would send for adding a design document to Apertis. This template
385 for the first email contains the description of the design document instead of
386 the design document itself. The idea is to promote involving the Apertis team
387 as early as possible, and ideally before completing the work.

388 The rationale for this approach is that it is very difficult for an external con-
389 tributor to understand the impact a contribution can bring to Apertis, and by
390 asking early, the work can be done in ways that are compatible with Apertis
391 and welcome by the Apertis team.

392 From: Your name <your email>

393 To: devel@lists.apertis.org
394 Subject: Robot Framework design document
395
396 Hi,
397
398 I want to contribute to Apertis, and I am sending this email to ask if our
399 proposal can be added to Apertis. I am sending the email based on the
400 contribution template I found on the Apertis website, and we are looking
401 forward for receiving feedback from the Apertis team.
402
403 Thank you,
404
405 Your name
406
407 -- // --
408
409 1. Me and my team
410 I am a developer, I am specialized in embedded devices, and I work in a product
411 team that creates IoT devices with all sorts of environmental sensors and
412 actuators.
413
414
415 2. What is the goal of my proposal
416 My proposal is for a design document that describes tools and workflows for
417 process automation using the Robot Framework. The Robot Framework is a generic
418 open source automation framework that can be used for automation of tests and
419 processes.
420
421 - From our perspective this adds value to the Apertis Universe. Do you agree?
422
423
424 2. State-of-the-art
425 We prefer the Robot Framework because it is mature, it is simple to use, and
426 because it has an active development community.
427
428 While there are other automation frameworks available, they tend to be purpose
429 specific. Examples of purpose specific automation frameworks that we considered
430 include Selenium and JUnit.
431
432 3. How does our contribution works?
433 The Robot framework has a layered architecture. The top layer is the simple,
434 powerful, and extensible keyword-driven descriptive language for testing and
435 automation. This language resembles a natural language, is quick to develop, is
436 easy to reuse, and is easy to extend. On the bottom layer of the architecture is
437 the item to be tested, or the process to be automated.
438

439 The middle layer is what makes the Robot Framework extensible: libraries. A
440 library, in Robot Framework terminology, extends the Robot Framework language
441 with new keywords, and provides the implementation for these new keywords. Each
442 Robot Framework library acts as glue between the high level language and low
443 level details of the item being tested, or of the environment in which the item
444 to be tested is present.

445

446

447 4. Potential impact on Apertis?

448 We are aware there the architecture of the Robot Framework is different from the
449 Architecture of LAVA. In some cases the Robot Framework accepts human
450 intervention with tests while LAVA expects everything to be automated. While we do
451 not fully understand to which extent this will impact Apertis, we expect that for our
452 design proposal will need to adapt to Apertis and LAVA constraints. Can you help us
453 here?

454

455 5. Benefits for Apertis?

456 The Robot Framework project is active for many years and is used for a variety
457 of use cases. We expect that adding the Robot Framework to the Apertis Universe
458 will bring Robot Framework users to Apertis.

459

460

461 6. What is the license of the main components?

462 The Robot Framework itself is licensed under the Apache License 2.0, however
463 Robot Framework libraries can use different licenses.

464

465

466 7. The plan to integrate the design into Apertis

467 Our understanding is that Apertis currently uses LAVA for testing, and that
468 images being tested are as close to production images as possible (almost no
469 testing instrumentation included). We propose to develop and/or modify a few
470 Robot Framework libraries, and to create a run-time compatibility layer for LAVA.
471 We expect that the combination of custom libraries with the run-
472 time compatibility
473 layer for LAVA will enable us to keep testing environments as close as possible
474 to production environments, and to adapt the execution of Robot Framework tests
475 to suit the Apertis and LAVA constraints.

476

477

478 8. Estimated work to implement the design

479 Our ballpark estimation to add or modify Robot Framework libraries and to create
480 the run-time compatibility layer for LAVA is of approximatedly 1500 hours of
481 work. But we need your help to fully understand the impact on the Apertis side.

482

483

484 9. High level implementation plan

485 While we understand our use case and requirements, we would like to receive
486 feedback from other potential users as soon as possible. Our idea is to deploy
487 the Robot Framework in stages to allow early involvement of other users:
488
489 - Add Robot Framework to the Apertis SDK to enable developers to use the Robot
490 Framework locally
491
492 - Robot Framework Integration development: Adapt libraries and create the run-
493 time
494 compatibility layer for LAVA
495
496 - Deployment on the Apertis infrastructure