



Upstreaming

# 1 Contents

2 **Where to upstream** **3**

3 **What can be upstreamed** **4**

4 Upstreaming changes made to a piece of Open Source software provides distinct  
5 advantages for the author of the changes and the ecosystem of users of software.  
6 The author can expect to see:

- 7 • Reduced overhead in on-going maintenance of their code base: With the  
8 changes available in the upstream version, the author will no longer need  
9 to port changes when upgrading to a newer version of the software.
- 10 • Reduced risk of incompatible changes and/or features being added to the  
11 upstream code base: When making local modifications to a code base,  
12 there is a risk that at some future point any local changes will fail to apply  
13 without significant changes or a comparable feature will be implemented  
14 with different semantics requiring either conversion to this feature to be  
15 carried out or continuing to carry the local change with much reduced  
16 advantages.
- 17 • Greater review of the proposed changes: Open source projects tend to  
18 review changes before they are merged and whilst not perfect, such reviews  
19 tend to be carried out by developers with a good working knowledge of  
20 the code base, which results in a better review than would be achieved in  
21 many settings.
- 22 • Potentially increased testing of added features: Other users of the software  
23 either evaluating or using the upstreamed features may uncover bugs or  
24 security holes in the added features that might otherwise go missed. This  
25 results in increased robustness for your users.
- 26 • Potential for further complementary features being added: The addition  
27 of a feature may prompt other developers to add complimentary features  
28 that prove useful to either you or your users.

29 The upstream project obviously benefits from the addition of features as this  
30 makes the software appealing to a wider audience and thus is likely to increase  
31 adoption. Having an active community may also help to increase adoption in  
32 its own right.

33 Whilst there is obviously an advantages for upstream projects to accept new  
34 features, it is also important that they thoroughly review and consider such  
35 changes. Bad changes could lead to instability of the application, negatively  
36 impacting all users of the project. Additionally the [maintainers](#)<sup>1</sup> are taking  
37 on the task of providing some maintenance for the added features. It is thus  
38 important that they ensure such changes comply with the coding conventions  
39 and other polices to ensure ease of maintenance and continued good health of  
40 the project.

---

<sup>1</sup><https://www.apertis.org/policies/contributions/#the-role-of-maintainers>

## 41 **Where to upstream**

42 As mentioned elsewhere, Apertis is a derivative of Debian, which it's self pack-  
43 ages software projects from many sources outside of Debian its self. Depending  
44 on the changes being made, this may present 3 options as to where to upstream  
45 the changes:

- 46 • Apertis
- 47 • Debian
- 48 • Main project of the software component

49 The effort required to get changes accepted by each of these places and the  
50 associated delay in seeing your changes reach a released version of Apertis are  
51 going to differ, often quite drastically and is frequently inversely proportional  
52 to the on-going maintenance costs.

53 As a user of Apertis, it is likely that submitting changes to Apertis may offer  
54 the lowest barrier to entry and fastest route to see the changes reflected in  
55 an Apertis release. There may be instances where Apertis offers the only real  
56 option, for example where Apertis is maintaining an older version of the project  
57 for licensing reasons.

58 It is likely that Debian will only accept very limited types of change. Some  
59 changes such as security and bug fixes may be more viable for upstreaming to  
60 Debian, as a general rule feature additions may be less likely to be accepted,  
61 though this will depend on how “alive”the upstream project is, what the feature  
62 is and how the maintainer feels about it (after all, the maintainer will be taking  
63 on the burden of maintaining the patches). Any patches that are accepted by  
64 Debian may take longer to be picked up by Apertis (depending on exactly where  
65 the changes landed).

66 The last option is to submit changes directly to the upstream project. Clearly  
67 packaging related changes can't be submitted here as these are not generally  
68 handled by the upstream project. It is also possible that the version used in the  
69 current Apertis is behind upstream development branch because Apertis prior-  
70 itizes stability over new features. However the upstream development branch  
71 is where changes would need to be submitted, even when using this branch  
72 incurs additional effort to port and test. The advantage to submitting to the  
73 upstream project is that the changes will require no further ongoing porting to  
74 newer versions as it will be in the main code base.

75 In order to alleviate the significant delay that a user of Apertis is likely to see  
76 between upstreaming to either Debian or upstream projects and the changes  
77 appearing in Apertis, it is very likely that Apertis would be willing to accept  
78 backported upstream changes to the version currently in use. This provides  
79 the user with the advantage of being able to immediately use the functionality  
80 without needing to carry local changes, whilst the Apertis developers can expect  
81 to only need to carry the changes in the short to medium term until the changes  
82 filter through.

## 83 What can be upstreamed

84 Most systems are comprised of parts that exist to provide a relatively standard  
85 environment that are likely to be (or could be) common to many devices using  
86 similar hardware or requiring similar functionality and parts that provide some  
87 kind of unique experience or custom logic specific to the device in question.  
88 It is the parts that form the standard environment where the advantages of  
89 upstreaming are most commonly exploited as these are the parts most likely to  
90 benefit others and which benefit the most from others usage.

91 Examples of items that would be prime candidates for upstreaming include:

- 92 • Drivers for publicly available devices, including peripheral devices and  
93 architectures/SoCs previously not supported
- 94 • Previously unsupported functionality in provided by supported devices
- 95 • Extending functionality for widely usable use cases in user space libraries  
96 and applications
- 97 • Bug fixes and security fixes for any upstream component

98 Ideally, such additions would be submitted to the main project in the first in-  
99 stance (with a backport submitted to Apertis once accepted upstream). Should  
100 upstream submission fail such patches would be considered on a case-by-case  
101 basis for addition into Apertis.

102 **Note:** Upstreaming is generally a process best considered from the outset. If  
103 upstreaming is planned at an early stage consider actively [working with the](#)  
104 [community](#)<sup>2</sup> during development, as this may streamline and simplify the devel-  
105 opment and upstreaming process.

106 Modifications and functionality that are not suitable for upstreaming will be  
107 considered on a [case by case basis](#)<sup>3</sup>. Whether they will be considered suitable  
108 for integration into the main Apertis repositories will depend in part on how  
109 broad the usefulness of the changes will be with the Apertis user base. At a  
110 minimum it would be necessary for such changes to comply with the coding  
111 standards of the relevant package, not impact the operation of Apertis for other  
112 users and be suitably licensed.

---

<sup>2</sup><https://www.apertis.org/policies/contributions/#upstream-early-upstream-often>

<sup>3</sup><https://www.apertis.org/policies/contributions/#adding-components-to-apertis>