



Supported API

# Contents

2	<b>New releases and API stability</b>	<b>2</b>
3	<b>API and ABI stability strategies</b>	<b>3</b>
4	The Android approach . . . . .	3
5	The iOS approach . . . . .	4
6	The Apertis/OpenSource approach . . . . .	4
7	The role of limiting the supported API surface . . . . .	6
8	How would incompatible changes impact the product and how to handle them? . . . . .	6
9	The GTK+ upgrade and a Clutter API break . . . . .	6
10	When a core library breaks . . . . .	7
11	When a “leaf” library breaks ABI . . . . .	8
12	ABI is not just library symbols . . . . .	8
13	The move to Wayland . . . . .	9
14	The GTK+ and Clutter merger . . . . .	9
15		
16	<b>API Support levels</b>	<b>9</b>
17	Custom APIs . . . . .	10
18	Enabling APIs . . . . .	11
19	OS APIs . . . . .	12
20	Internal APIs . . . . .	12
21	External APIs . . . . .	12
22	Differing stability levels . . . . .	13
23	Maintaining API stability . . . . .	14
24	<b>Components</b>	<b>14</b>
25	<b>Conclusion</b>	<b>15</b>
26	The goal of this document is to explain the relevant issues around API (Application Programming Interface) and ABI (Application Binary Interface) stability and to make explicit the APIs and ABIs that can be and will be guaranteed to be available in the platform for application development.	
27		
28		
29		
30	It will be explained as well how we are going to deal with situations where certain components break their API/ABI.	
31		

## New releases and API stability

Software systems are typically composed of several components with some depending on others. Components need to make assumptions about how their dependencies behave, in order to use them. These assumptions are categorized in API and ABI depending on whether they are resolved at build time or at run-time, respectively. As components evolve over time and their behavior changes, so may their API and ABI.

39 In systems composed of thousands of components, each time a component  
40 changes, potentially hundreds of other components could break. Fixing each  
41 of those components could cause other breaks in turn. Without a way to man-  
42 age those changes, assembling and maintaining non-trivial systems wouldn't be  
43 a practical enterprise.

44 To manage this complexity, components which are to be depended upon by  
45 others set an API/ABI stability policy. This policy states under which cir-  
46 cumstances new releases can be expected to break API or ABI. This allows  
47 the system integrator to update to newer releases of components with some as-  
48 surance that other components won't break as a result. These guarantees also  
49 allow new releases of components to simply depend upon the last "known-good"  
50 release of each of their dependencies instead of requiring them to be constantly  
51 tested against newer dependencies.

52 Most components will keep stable branches in which API - and often ABI -  
53 are not allowed to break, and normally only bug fixes and minor features will  
54 be merged into these branches. It is generally recommended that components  
55 (particularly, stable ones) depend only on stable branches of their dependencies.  
56 Releases in a stable branch are referred to as "backwards compatible" because  
57 components that depend upon a given release will continue to work with later  
58 releases in that same branch.

59 By libraries keeping API stability in stable branches and by libraries and appli-  
60 cations depending on stable versions of libraries, breaks are greatly reduced to  
61 manageable levels.

62 An API can consist of multiple parts: for a typical C library, the API will  
63 be the C function and type declarations, plus the gobject-introspection (GIR)  
64 description of the API. Similarly, an ABI can consist of multiple parts: the C  
65 function and type declarations, plus the D-Bus API for a system service, for  
66 example.

67 The GIR API is especially relevant for further development of Apertis, as it is  
68 planned to allow apps to be written in non-C languages such as JavaScript. In  
69 this situation, API stability requires both the C declarations to be stable, plus  
70 the conversion of those declarations to a GIR file to be stable — so it is affected  
71 by changes in the implementation of the GIR scanner (the g-ir-scanner utility  
72 provided by gobject-introspection). This is covered further in **ABI is not just**  
73 **library symbols**.

## 74 **API and ABI stability strategies**

75 There is a tension between keeping the development environment stable and  
76 keeping up with novelties. Following is an investigation about how various mo-  
77 bile platforms have tackled this issue that hopefully provides enough information  
78 for a practical strategic decision on how to handle that tension.

## 79 The Android approach

80 Android makes a promise of forward-compatibility for the main Android APIs.  
81 Although Android has been built on top of Linux and using a Java virtual  
82 machine, no APIs of these platforms are considered to be part of the Android  
83 platform.

84 Instead of reusing existing components and libraries Google decided to write  
85 almost everything from scratch, including a C library, a graphics subsystem,  
86 audio, web and multimedia subsystems and APIs.

87 This approach has the big disadvantage of not reusing and sharing much of the  
88 work done by the open source community in similar projects, which means a  
89 significant investment and hundreds of thousands of hours of engineering time  
90 spent building and maintaining everything. On the plus side, those APIs and  
91 the underlying components they are built upon are fully controlled by Google,  
92 and submit to whatever requirements the Android platform has, giving Google  
93 full control regarding tilting the balance in favour of stability or break-through  
94 as it sees fit.

95 Although Google has been very successful in keeping its API/ABI stability  
96 promises, it has made incompatible changes in almost every release. From API  
97 level 13 to 14 (in other words, from Android 3.2 to 4.0) alone there were a few  
98 dozen API deprecations and *removals*<sup>1</sup>, including methods, class and interface  
99 fields, and so on. Each new version brings in its release notes a report of API  
100 differences compared to the last version. In addition to these, underlying compo-  
101 nent changes have caused applications to misbehave and crash when assuming  
102 a certain behaviour that got changed.

## 103 The iOS approach

104 Apple has been known for wanting to control every bit of the products they  
105 make. From hardware all the way to third-party application design, Apple tends  
106 to influence or enforce its own rules. The iOS is no exception: instead of reusing  
107 existing open source APIs, Apple designed and built their own components and  
108 APIs from the ground up. The same disadvantages Android's approach has are  
109 also present here: instead of sharing the cost of building all of the basic tools  
110 with lots of developers world wide, Apple decided to build everything itself,  
111 making a significant investment in terms of money and engineering time.

112 The main difference between Android and iOS, though, are that Apple did not  
113 have to start from scratch: they had Mac OS X already, and were able to  
114 reuse some of the work they have done previously, although that itself brings  
115 a disadvantage: the need to balance the needs of the desktop use case and  
116 the mobile use case in a single code base. The advantages, though, are the  
117 same: Apple is fully in control of the system from the ground up, and can make  
118 decisions on tilting the balance between stability and break-through.

---

<sup>1</sup>[http://developer.android.com/sdk/api\\_diff/14/changes/alldiffs\\_index\\_removals.html](http://developer.android.com/sdk/api_diff/14/changes/alldiffs_index_removals.html)

119 Apple, like Google, has also been successful keeping compatibility, but has had  
120 its set of incompatible changes in every release. The [API changes between iOS](#)  
121 [4.3 to 5](#)<sup>2</sup>, for instance, has a couple tens of *removed or renamed* classes, fields  
122 and methods.

## 123 The Apertis/OpenSource approach

124 Open source projects like GNOME have been very successful at providing bal-  
125 ance to the tension by having API/ABI stability promises, but as the need  
126 for technology overhauls appeared, keeping backwards compatibility has often  
127 proven very costly, and a choice to break compatibility and refresh the platform  
128 has been made.

129 That was the case, for instance, with the release of GNOME3. The GNOME  
130 project had to some extent maintained compatibility with applications that were  
131 written all the way back in 2002, and had accumulated a considerable amount  
132 of deprecated functionality and APIs that burdened the project, slowing down  
133 progress and requiring a lot of maintenance work. Those had to be left behind  
134 the project in order to bring it up-to-date with the expectations of the current  
135 decade.

136 The big advantage of using open source components is most of the hard work of  
137 building all of the pieces of infrastructure and even some applications has been  
138 made, leaving hardware integration, application development, customization,  
139 specific features and QA as the main required work before going to market,  
140 instead of having a much larger team that would build everything from scratch,  
141 or licensing a proprietary components.

142 The main disadvantage to this approach is that the decision on how to tilt  
143 the balance between stability and freshness is not under the full control of the  
144 company building the product: some decisions will be made by the projects that  
145 build the various components that make up the solution that can increase the  
146 cost of keeping stability while still maintaining freshness.

147 For instance: Google has full control of Android's underlying graphics stack,  
148 Surface Flinger, and is able to ensure its compatibility moving forward; it is  
149 also able to make APIs deal transparently with changes in this underlying layer.  
150 The same goes for Apple and its iOS. When it comes to the open source graph-  
151 ics stack, a move from the current Xorg infrastructure to the next-generation  
152 Wayland will break some of the underlying assumptions made by applications.

153 Some of the core libraries that are parts of the graphics stack are also likely to  
154 change, taking advantage of the API stability break imposed by the move to  
155 a new graphics infrastructure to also perform some changes to their core and  
156 APIs. Some projects may also decide to break their stability promises from time  
157 to time for technology overhauls, like GNOME did with GNOME 3. We will

---

<sup>2</sup><https://developer.apple.com/library/ios/#releasenotes/General/iOS50APIDiff/index.html>

investigate some theoretical and real world cases in order to get a more concrete example of how these overhauls may present themselves, and how they can be handled.

There are several options when dealing with backwards-incompatible novelties: delaying the integration of a new release, for instance, is the best way to guarantee stability, but that will only delay the impact of the changes. Building a set of APIs that abstract some of the platform can also be sensible: applications using high level widgets can be shielded from changes done at the lower levels – Clutter, Mx, and so on.

To conclude: taking advantage of open source code takes away some of the control over the platform’s future. While Google and Apple are able to decide exactly what happens to the components that make up Android and iOS in the future, someone basing their product on an open source platform doesn’t. It’s important to notice that that is also the case for companies building products based on Android, and maybe even more so: when Google decided that Android Honeycomb would not be released, many companies were left without the latest version of Android to base their products on.

Also, like GNOME, Windows and Mac OS have started afresh at some point in time, to be able to bring their products to the next level, it is very likely there will come a time in which iOS and Android will go through a similar major change on their foundations, and companies basing their products on Android will have to decide how to handle the upgrade, when it happens.

## The role of limiting the supported API surface

While the API and ABI promises made by Android and iOS have been largely successful, it is important to note that they do not cover everything an application may need. Core services like graphics and networking are covered, but more specific functionality is not. One example is JSON processing. JSON is one of the most widely used formats for exchanging data between apps and servers.

There are no APIs at all for this format in iOS. Applications that need to use JSON need to either roll their own implementation or embed a JSON processing library into their application. The same goes for APIs to access Youtube and other Google services through its GData protocol.

See < <http://www.appdevmag.com/10-ios-libraries-to-make-your-life-easier/%3E><sup>3</sup> for more examples of missing APIs and replacements that can be embedded

Android has similar limitations. Android devices are not guaranteed to have APIs for Google services, and although add-ons exist to bolt on those APIs, they cannot be redistributed, in some cases. For services that use GData, there

---

<sup>3</sup><http://www.appdevmag.com/10-ios-libraries-to-make-your-life-easier/%3E>

197 is also an add-on library that can be embedded in the application, but there are  
198 no API/ABI guarantees.

199 Imposing those limits on which APIs are guaranteed to not change (or change  
200 as little as possible in reality) makes it possible for Android and iOS to lower  
201 the maintenance costs for the platform, while making it possible to embed li-  
202 braries into applications allows applications to not be completely limited by the  
203 available standard APIs. Note also that embedded libraries can only be used  
204 by the application embedding it, avoiding inter-application dependencies. That  
205 is one of the reasons Collabora is suggesting that a set of libraries be specified  
206 to be handled as supported.

## 207 **How would incompatible changes impact the product and** 208 **how to handle them?**

209 This section aims at investigating some cases where a line was drawn and old  
210 APIs were left behind, and how products based on or simply shipping those  
211 APIs handled it. The arrival of GNOME 3 in early 2011 drew the line and  
212 allowed for the clean up of APIs that were almost 10 years old, with few or  
213 no forward compatibility breakages through that period. It provides a lot of  
214 insights at how to handle that kind of structural overhaul.

### 215 **The GTK+ upgrade and a Clutter API break**

216 GTK+ is the main toolkit used by the GNOME system. The upgrade to GTK+  
217 3.0 was very smooth, for such a big upgrade. Applications required changes,  
218 but not all applications needed to be ported at once, since everything that made  
219 up the library changed name, making it installable in parallel with GTK+ 2.  
220 This means simple applications written using the toolkit still work, even if you  
221 have GTK+ 3-based applications installed and working. So that is exactly how  
222 distributors handled the situation: both libraries are installed as long as there  
223 are applications that need the old one.

224 A very similar situation would surface if Clutter and Mx happened to break their  
225 API and ABI promises: applications that aren't updated to use the new APIs  
226 and ABIs would simply continue using the older Clutter and Mx libraries. An  
227 additional burden would appear for the teams designing higher level widgets,  
228 though: the widgets would have to be supported for both library versions,  
229 and care would need to be taken to not have an application link to the old  
230 Clutter/Mx and with the higher level widgets built with the new ones.

231 There are several facilities to make this possible available in the debian pack-  
232 aging tools used by the base distribution Apertis is built on, and also in the  
233 development tools used by those libraries. Provided they are used correctly this  
234 specific case should not prove too difficult. Most distributions that handled this  
235 kind of breakage spent a lot of time tuning dependencies and other package  
236 relationships, and making sure no interfaces other than the binary ones were in

237 disagreement, though. Some of the Collabora developers who are participating  
238 in the Apertis project are responsible for a significant part of the work that has  
239 been done to make the transition smooth in Debian. Their experience with it is  
240 that it is a very time consuming process, with many corner cases and subtleties  
241 to be taken care of, and even then several trade-offs had to be made.

## 242 **When a core library breaks**

243 Some applications are a bit special: most browser plugins, for instance, relied  
244 on the browser being written in GTK+ 2 – since that is what Firefox uses on  
245 Linux/X11. That is not a problem for a browser built in Qt, or Clutter, for  
246 instance, since they can look for the system GTK+ 2 library, open it and use  
247 its symbols to perform the initialization some plugins expect. It is a problem,  
248 though, for browsers written in GTK+ 3: as soon as the plugin is loaded there  
249 will be symbols from both GTK+ 3 and GTK+ 2 in the symbol resolution table,  
250 and that will lead to subtle and hard to debug bugs, and to crashes. That is  
251 one of the reasons why Firefox has decided to not move to GTK+ 3.

252 The same happens with GStreamer plugins. If a library is used by both a  
253 GStreamer plugin and an application, and that library changes the same prob-  
254 lem described for browser plugins would happen. That would be the case if, for  
255 instance, an application uses clutter-gst – since the application and the clutter-  
256 gst video sink both link to Clutter, they would need to be linked to the same  
257 version of the library to work properly.

258 Plugins are not the only case in which such problems happen. If a core library  
259 like glib breaks compatibility similar issues will appear for all of the platform.  
260 Almost every application links to glib and so do many libraries, including core  
261 ones like Clutter. If a new version of glib is released which breaks ABI, all of  
262 these would have to be migrated to the new library at once, otherwise symbol  
263 clashes like the ones described above would happen. In GNOME 3 glib has  
264 not broken compatibility, but it is expected to break it at some point in the  
265 (medium term) future.

266 As discussed in the previous section, ensuring forward compatibility after such a  
267 break in the ABI of glib would only be possible with a very significant effort, and  
268 might prove to not be viable. Collabora would recommend that turning points  
269 like this be treated as major upgrades to the platform, requiring applications to  
270 be reworked. Such upgrades can be delayed by a few releases to allow enough  
271 time for the applications to be updated, though.

## 272 **When a “leaf” library breaks ABI**

273 When a core library such as glib breaks, the impact will be felt throughout the  
274 platform, but when a library that is used only by a few components breaks there  
275 is more room for adjustment. It’s unlikely that both libraries and applications  
276 would link to libstartup-notification, for instance. In such cases the new version



277 of the library can be shipped along with the old one, and the old one can be  
278 maintained for as long as necessary.

### 279 **ABI is not just library symbols**

280 A leaf library may end up causing more issues, though, if it breaks. GNOME  
281 3 has provided us with an example of that: the GNOME keyring is GNOME's  
282 password storage. It's made up of a daemon (that among other things provides  
283 a D-Bus service), and a client library for applications to use. GNOME keyring  
284 has undergone a change in the protocol, and both the library and the daemon  
285 were updated. The library was parallel installable with the old one, but the new  
286 daemon completely replaced the old one.

287 But the old client library and the new daemon did not know how to talk to each  
288 other, so even though applications would not crash because of a missing library  
289 or missing symbols, they were not able to store or obtain passwords from the  
290 keyring. That is also what would happen in case a D-Bus service changes its  
291 interface.

292 In case something like this happens it is possible to work around the issue by  
293 adding code to the daemon to keep supporting the old protocol/interface, but  
294 this increases the maintenance burden and the cost/benefits ratio needs to be  
295 properly assessed, since it may be significant.

296 Similarly, the GIR interface for a library forms part of its public API. The GIR  
297 interface is a high-level, language-agnostic API which maps directly to the C  
298 API, and can be used by multiple language bindings to automatically allow the  
299 library to be used from those languages. Its stability depends on the stability of  
300 the underlying C library, plus the stability of the GIR generation, implemented  
301 by g-ir-scanner.

### 302 **The move to Wayland**

303 Moving to Wayland is a fairly big change, but the impact on application com-  
304 patibility may not be that big. If applications are using only standard Clutter  
305 and Mx APIs (or higher level APIs built on top of them) they would just work.  
306 If the application relies on something related to X, though, and uses any of the  
307 Clutter X11 functions, then that will require that they be ported.

308 That is a good reason for making those APIs part of the unsupported set, and  
309 if necessary provide APIs as part of the higher level toolkit to accommodate  
310 application needs. Wayland will allow an X server to be run and paint to one  
311 of its windows, so extreme cases could be handled by using that feature, but  
312 relying on it may prove unwise.

### 313 **The GTK+ and Clutter merger**

314 There has been discussion among GNOME developers recently about merging  
315 Clutter and GTK+ into a single toolkit. GTK+ is a powerful toolkit with many

316 years of experience built in, and solving many of the problems posed by complex  
317 UIs, but it lacks the eye candy and some of the features people now expect in a  
318 modern toolkit. Clutter on the other hand has all of the eye candy and features  
319 one expects from a modern toolkit, but lacks the toolkit part. While Mx and St,  
320 the GNOME Shell's toolkit, do provide some widgets and higher level features,  
321 they are not nearly as fully featured and mature as GTK+. The existence of  
322 so many toolkits is being seen as fragmentation of the developer story in the  
323 GNOME platform, which also plays a role in these discussions.

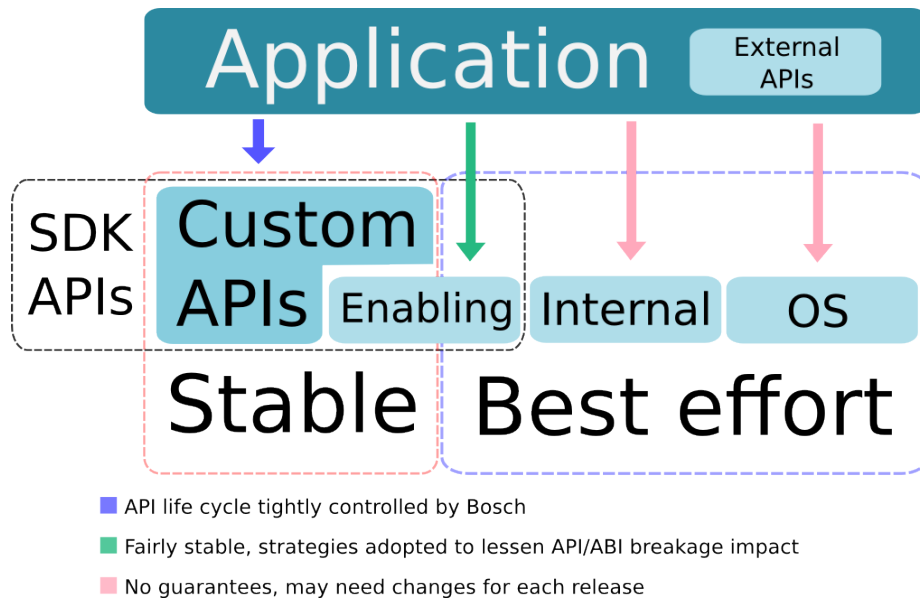
324 When the merger of Clutter and GTK+ happens, the impact and solutions  
325 would be pretty much the same as if Clutter and Mx break ABI. Old libraries  
326 and applications using Clutter and Mx would remain working, but care would  
327 have to be exercised in making sure no process ends up using the two versions at  
328 the same time. It would also lead the project to making a decision on whether  
329 to rebase the higher level widgets on the new GTK+ 4 (as the merged library  
330 is called in discussions) or not.

331 According to the maintainers, Mx is still in use by Intel in some of their appli-  
332 cations and will be used for the netbook UI in Tizen, so its medium-term future  
333 appears to be fairly certain at this point.

## 334 **API Support levels**

335 A number of API support levels has been indicated recognizing that some bits  
336 of the platform are more prone to change than others, and given the strategy  
337 of building higher level custom APIs. The custom and enabling APIs make up  
338 what is often called the SDK APIs. They are the ones with better promises,  
339 and for which Collabora will try to provide smooth upgrade paths when changes  
340 come about, while the APIs on the lower levels will not get as much work, and  
341 application developers will be made aware that using them means the app might  
342 need to be updated for a platform upgrade.

343 The overall strategy being considered right now to assign APIs to each of these  
344 support levels is to start with the minimum set of libraries required to run the  
345 Apertis system being part of the image with all libraries assigned to the Internal  
346 APIs support level, and gradually promote them as development progresses and  
347 decisions are made. The following sections describe the support levels.



348

### 349 Custom APIs

350 The Custom APIs are high level APIs built on top of the middleware provided by  
 351 Collabora. These APIs do not expose objects, types or data from the underlying  
 352 libraries, thus providing easier and abstract ways of working with the system.

353 Examples of such APIs are the share functionality, and a number of UI com-  
 354 ponents that have been designed and built for the platform. Collabora has  
 355 had only limited information about these components, so an assessment of how  
 356 effectively they shield store applications from lower support level libraries is  
 357 currently not possible.

358 For these components to deliver on their promise of abstracting the lower level  
 359 APIs it is imperative that they expose no objects, data types, functions and so  
 360 on from other libraries to the application developer. Collabora will be ready  
 361 to assist on defining and refining the Custom APIs to cover basic needs for  
 362 applications.

### 363 Enabling APIs

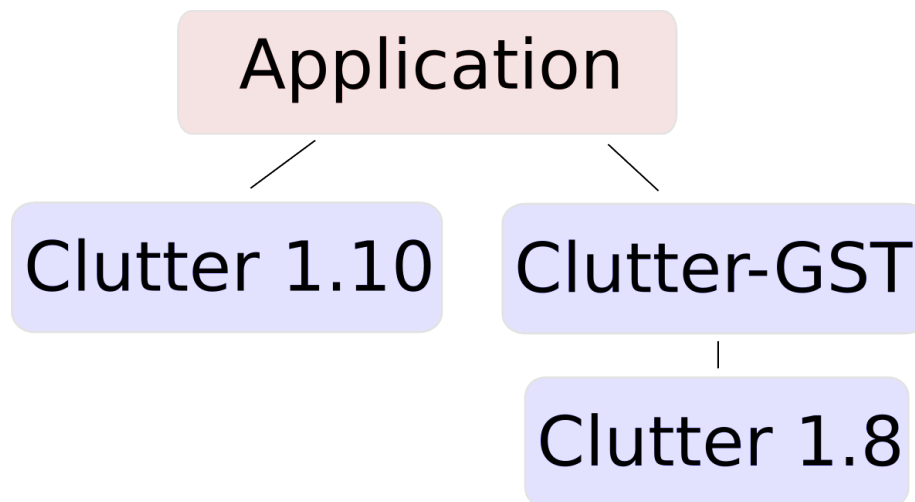
364 These APIs are not guaranteed to be stable between platform upgrades, but  
 365 work may be done on a case-by-case basis to provide a smooth migration path,  
 366 with old versions coexisting with newer ones when possible. Most existing open  
 367 source APIs related to core functionality fall in this support level: Mx, clutter,  
 368 clutter-gst, GStreamer, and so on.

369 As discussed in section 3.5.1, **The GTK upgrade and a Clutter API break**,  
 370 there are ways to deal with ABI/API breakage in these libraries. Keeping both

371 versions installed for a while is one of them. In the short term there will be at  
372 least one set of API changes that will have a big impact on the Apertis project:  
373 [Clutter 2.0](#)<sup>4</sup>. That new version of clutter is one of the steps in preparation for  
374 a future merge of GTK+ and Clutter.

375 It is possible that this new version of Clutter is released while the Apertis project  
376 is still not far enough in development that a switch can be made. However, in  
377 case that is not possible, a plan will need to be laid out to properly migrate  
378 to this new version in a future release. Being based on Clutter, the main SDK  
379 APIs that relate to UI will need to be ported, of course. Components that are  
380 based on Clutter such as clutter-gst will need to be updated too. Illustration  
381 Illustration shows how an application process could end up in this situation.

382 This would lead to the kind of problems discussed in [When a core library breaks](#)  
383 for applications that use clutter both directly and indirectly through another  
384 library that uses clutter under the hood, for instance. An application that uses  
385 both SDK UI APIs and an earlier version of clutter would have to be updated.  
386 An application which relies solely on Clutter would still work fine by just having  
387 the old version of clutter around. The same would apply to an application which  
388 relies solely on the SDK UI APIs, of course.



389

## 390 OS APIs

391 The OS APIs include low level libraries such as glib and its siblings gio, gdbus,  
392 as well as system services such as PulseAudio, glibc and the kernel. Applications  
393 reaching down to these components would, as is the case for enabling APIs, not  
394 necessarily work without changes after a platform upgrade.

<sup>4</sup><http://wiki.clutter-project.org/wiki/ClutterChanges:2.0>

## 395 Internal APIs

396 These are APIs used to build the Apertis system itself but not exposed to store  
397 applications. A library might get assigned to this support level if it is required  
398 to implement system features, but its API is too unstable to expose to from-  
399 store applications. Some libraries that fit this support level might also be in the  
400 External APIs one.

## 401 External APIs

402 Some libraries are not core enough that they warrant being shipped along with  
403 the main system or are not very stable API-wise. One such example is poppler,  
404 which changes API and ABI fairly often and is not really required for most  
405 applications – it will certainly be used on the main PDF viewing application,  
406 and most other applications will simply yield to the system viewer when faced  
407 with a PDF file.

408 That means poppler is a good candidate for bundling with the applications that  
409 need it instead of being part of the core supported APIs.

## 410 Differing stability levels

411 While the Enabling, Custom, External, Internal and OS categories separate  
412 APIs based on the level of control and direct involvement we have over them,  
413 a separate dimension is needed to track the stability of APIs, with four levels:  
414 private, unstable, stable, and deprecated. An API starts as private, and can  
415 transition to any of the other levels. Transitions between stable and deprecated  
416 are possible, but an API can never change or go back to being unstable or  
417 private once it is stable — this is one of the stability guarantees.

418 It may be possible to move a library from the unstable level to the stable level  
419 piecewise, for example by initially exposing a limited set of core functions as  
420 stable, while marking the rest of the API as ‘currently unstable’. Old API could  
421 later be marked as deprecated. Further, it may be desirable to expose the same  
422 API at different levels for different languages. For example, a library might be  
423 stable for the C language, but unstable when used from JavaScript, pending  
424 further testing and documentation work to mark it as stable.

425 This approach allows a phased introduction of stable APIs, giving sufficient  
426 time for them to be thoroughly reviewed and tested before committing to their  
427 stability.

428 This could be implemented in the GIR files for an API, with annotations ex-  
429 tracted from the gtk-doc comments of the API’s C source code — gtk-doc  
430 currently supports a ‘Stability’ annotation. As an XML format, GIR is exten-  
431 sible, and custom attributes could be used to annotate each function and type  
432 in an API with its stability, extracted from the gtk-doc comments. Separate  
433 documentation manuals could then be generated for the different stability lev-

434 els, by making small modifications to the documentation generation utilities in  
435 gtk-doc.

436 Restricting less stable or deprecated parts of an API from being used by an  
437 app written in C is technically complex, and would likely involve compiling two  
438 versions of each library. It is suggested that less stable functions and types are  
439 always exposed, with the understanding that app developers use them at their  
440 own risk of having to keep up with API-incompatible changes between Apertis  
441 versions. Their existence would not be obvious, as they would not be included  
442 in the documentation for the stable API.

443 By contrast, restricting the use of such APIs from high-level languages is simpler:  
444 as all language bindings use GIR, only the GIR files and the infrastructure  
445 which handles them needs modifying to support varying the visibility of APIs  
446 according to their stability level. The bindings infrastructure already supports  
447 ‘skipping’ specific APIs, but this is not currently hooked up to their advertised  
448 stability. A small amount of work would be needed to enable that.

## 449 **Maintaining API stability**

450 It is easy to accidentally break API or ABI stability between releases of a library,  
451 and once a release has been made with an API break, that break cannot be  
452 undone.

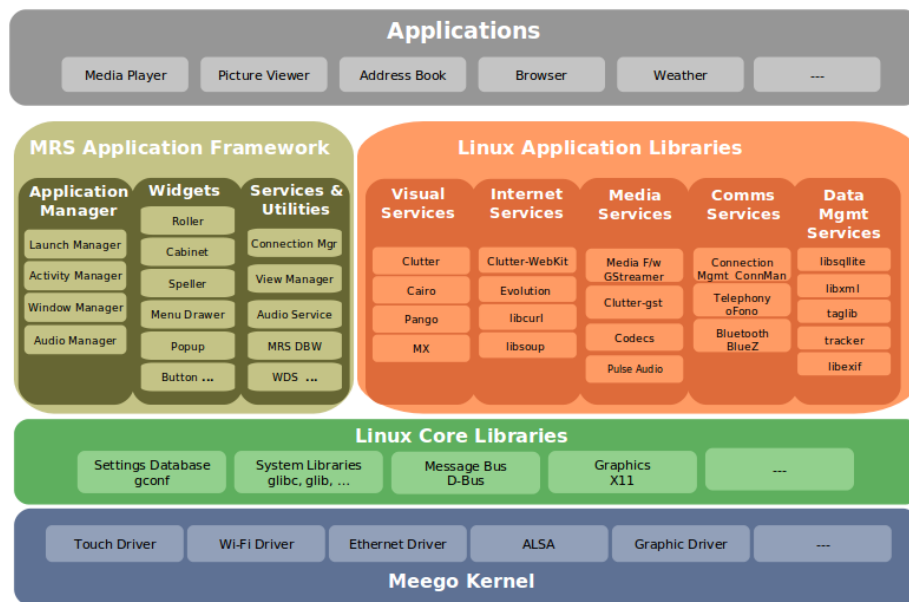
453 The Debian project has some tooling to detect API and ABI changes between  
454 releases of a library, though this is invoked at packaging time, which is after the  
455 library has been officially released and hence after the damage is done.

456 This tooling could be leveraged to perform the ABI checks before making a  
457 library release.

458 While such tools exist for C APIs, no equivalents exist for GIR and D-Bus  
459 APIs; the stability of these must currently be checked manually for each release.  
460 As both APIs are described using XML formats, developing tools for checking  
461 stability of such APIs would not be difficult, and may be a prudent investment.

## 462 **Components**

463 To illustrate how the platform APIs relate to Apertis-specific APIs, we are  
464 reproducing here a diagram taken from the Apertis SDK documentation. The  
465 components listed in the table below belong to the orange and green boxes:



466

467 The following table has a list of libraries that are likely to be on Apertis images  
 468 or fit into one of the supported levels discussed before. The table has links  
 469 to documentation and comments on API/ABI stability promises made by each  
 470 project for reference. As discussed before, fitting components into one of the  
 471 supported levels will be an iterative process throughout development, so this  
 472 table should not be seen as a canonical list of supported APIs.

Name	Version	API reference
GLibc	2.14	<a href="http://www.gnu.org/software/libc/manual/html_node/index.html">http://www.gnu.org/software/libc/manual/html_node/index.html</a>
OpenGL ES	2.0	<a href="http://www.khronos.org/opengles/sdk/docs/man/">http://www.khronos.org/opengles/sdk/docs/man/</a>
EGL	1.4	<a href="http://www.khronos.org/registry/egl/specs/eglspec.1.4.20110406.pdf">http://www.khronos.org/registry/egl/specs/eglspec.1.4.20110406.pdf</a>
GLib	2.32	<a href="http://developer.gnome.org/glib/2.31/">http://developer.gnome.org/glib/2.31/</a>
Cairo	1.10	<a href="http://cairographics.org/documentation/">http://cairographics.org/documentation/</a>
Pango	1.29	<a href="http://developer.gnome.org/pango/stable/">http://developer.gnome.org/pango/stable/</a>
Cogl	1.10	<a href="http://docs.clutter-project.org/docs/cogl/unstable/">http://docs.clutter-project.org/docs/cogl/unstable/</a>
Clutter	1.10	<a href="http://docs.clutter-project.org/docs/clutter/unstable/">http://docs.clutter-project.org/docs/clutter/unstable/</a>
Mx	1.4	<a href="http://docs.clutter-project.org/docs/mx/stable/">http://docs.clutter-project.org/docs/mx/stable/</a>
GStreamer	1.0	<a href="http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html">http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html</a>
Clutter-GStreamer	1.6	<a href="http://docs.clutter-project.org/docs/clutter-gst/stable/">http://docs.clutter-project.org/docs/clutter-gst/stable/</a>
GeoClue	0.12	<a href="http://www.freedesktop.org/wiki/Software/GeoClue">http://www.freedesktop.org/wiki/Software/GeoClue</a>
LibXML2	2.7	<a href="http://xmlsoft.org/html/index.html">http://xmlsoft.org/html/index.html</a>
libsoup	2.4	<a href="http://developer.gnome.org/libsoup/unstable/">http://developer.gnome.org/libsoup/unstable/</a>
librest	0.7	<a href="http://developer.gnome.org/librest/unstable/">http://developer.gnome.org/librest/unstable/</a>
libchamplain	0.14.x	<a href="http://developer.gnome.org/libchamplain/unstable/">http://developer.gnome.org/libchamplain/unstable/</a>
Mutter	3.3	
ConnMan	0.78	<a href="http://git.kernel.org/?p=network/connman/connman.git;a=tree;f=doc;hb=">http://git.kernel.org/?p=network/connman/connman.git;a=tree;f=doc;hb=</a>

Name	Version	API reference
Telepathy-GLib	0.18	<a href="http://telepathy.freedesktop.org/doc/telepathy-glib/">http://telepathy.freedesktop.org/doc/telepathy-glib/</a>
Telepathy-Logger	0.2	<a href="http://telepathy.freedesktop.org/doc/telepathy-glib/">http://telepathy.freedesktop.org/doc/telepathy-glib/</a>
Folks	0.6	<a href="http://telepathy.freedesktop.org/doc/folks/c/">http://telepathy.freedesktop.org/doc/folks/c/</a>
PulseAudio	1.1	<a href="http://freedesktop.org/software/pulseaudio/doxygen/">http://freedesktop.org/software/pulseaudio/doxygen/</a>
Bluez	4.98	<a href="http://git.kernel.org/?p=bluetooth/bluez.git;a=tree;f=doc">http://git.kernel.org/?p=bluetooth/bluez.git;a=tree;f=doc</a>
libstartup-notification	0.12	See Notes
libecal	3.3	<a href="http://developer.gnome.org/libecal/3.3/">http://developer.gnome.org/libecal/3.3/</a>
SyncEvolution	1.2	<a href="http://api.syncevolution.org/">http://api.syncevolution.org/</a>
GUPnP	0.18	<a href="http://gupnp.org/docs">http://gupnp.org/docs</a>
libGData	0.11	<a href="http://developer.gnome.org/gdata/unstable/">http://developer.gnome.org/gdata/unstable/</a>
Poppler	0.18	There is minimal inline API documentation
libsocialweb	0.26	GLib-based API has no documentation
Grilo	0.1	API docs in sources
Ofono	1.0	<a href="http://git.kernel.org/?p=network/ofono/ofono.git;a=tree;f=doc">http://git.kernel.org/?p=network/ofono/ofono.git;a=tree;f=doc</a>
WebKit-Clutter	1.8.0	
libexif	0.6.20	<a href="http://libexif.sourceforge.net/api/">http://libexif.sourceforge.net/api/</a>
TagLib	1.7	<a href="http://developer.kde.org/~wheeler/taglib/api/index.html">http://developer.kde.org/~wheeler/taglib/api/index.html</a>

## Conclusion

Open Source has been chosen in order to be able to reuse code that is freely available and for its customization potential. It is also desired to keep the platform up-to-date with fresh new open source releases as they come about. While choosing to leverage Open Source software does lower cost and the required investment significantly, it does bring with it some challenges when compared to building everything and controlling the whole platform, especially when it comes to the tension between stability and novelty.

Those challenges will have to be met and worked upon on a case-by-case basis, and trade-offs will have to be made. Like other distributors of open source software have done over the years, delaying adoption of a particular technology or newer versions of a core package goes a long way in ensuring platform stability and providing safe and manageable upgrade paths, so it is certainly an option that must be considered. Other solutions should of course be considered and planned for, including shipping more versions of the same library in parallel. Limiting the API that is considered supported and requiring that some libraries be statically linked or be shipped along with the program are also tools that should be used where necessary.