



Supported API

1 **Contents**

2 **Supported API** **2**

3 Introduction 2

4 New releases and API stability 2

5 API and ABI stability strategies 3

6 The Android approach 3

7 The iOS approach 4

8 The ApERTIS/OpenSource approach 4

9 The role of limiting the supported API surface 6

10 How would incompatible changes impact the product and how to

11 handle them? 6

12 API Support levels 10

13 Custom APIs 10

14 Enabling APIs 11

15 OS APIs 12

16 Internal APIs 12

17 External APIs 12

18 Differing stability levels 12

19 Maintaining API stability 13

20 Components 14

21 Conclusion 15

22 **Supported API**

23 **Introduction**

24 The goal of this document is to explain the relevant issues around API (Applica-
25 tion Programming Interface) and ABI (Application Binary Interface) stability
26 and to make explicit the APIs and ABIs that can be and will be guaranteed to
27 be available in the platform for application development.

28 It will be explained as well how we are going to deal with situations where
29 certain components break their API/ABI.

30 **New releases and API stability**

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

37 In systems composed of thousands of components, each time a component
38 changes, potentially hundreds of other components could break. Fixing each

39 of those components could cause other breaks in turn. Without a way to man-
40 age those changes, assembling and maintaining non-trivial systems wouldn't be
41 a practical enterprise.

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

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

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

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

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

72 **API and ABI stability strategies**

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

77 **The Android approach**

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

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

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

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

101 **The iOS approach**

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

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

¹http://developer.android.com/sdk/api_diff/14/changes/alldiffs_index_removals.html

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

121 **The ApERTIS/OpenSource approach**

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

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

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

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

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

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

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

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

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

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

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

178 **The role of limiting the supported API surface**

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

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

189 See < [http://www.appdevmag.com/10-ios-libraries-to-make-](http://www.appdevmag.com/10-ios-libraries-to-make-your-life-easier/%3E)
190 [your-life-easier/%3E](http://www.appdevmag.com/10-ios-libraries-to-make-your-life-easier/%3E)³ for more examples of missing APIs and
191 replacements that can be embedded

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

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

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

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

205 **How would incompatible changes impact the product and how to** 206 **handle them?**

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

213 **The GTK+ upgrade and a Clutter API break**

214 GTK+ is the main toolkit used by the GNOME system. The upgrade to GTK+
215 3.0 was very smooth, for such a big upgrade. Applications required changes,
216 but not all applications needed to be ported at once, since everything that made
217 up the library changed name, making it installable in parallel with GTK+ 2.
218 This means simple applications written using the toolkit still work, even if you
219 have GTK+ 3-based applications installed and working. So that is exactly how
220 distributors handled the situation: both libraries are installed as long as there
221 are applications that need the old one.

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

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

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

240 **When a core library breaks**

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

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

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

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

270 **When a “leaf” library breaks ABI**

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

277 **ABI is not just library symbols**

278 A leaf library may end up causing more issues, though, if it breaks. GNOME
279 3 has provided us with an example of that: the GNOME keyring is GNOME's
280 password storage. It's made up of a daemon (that among other things provides
281 a D-Bus service), and a client library for applications to use. GNOME keyring
282 has undergone a change in the protocol, and both the library and the daemon
283 were updated. The library was parallel installable with the old one, but the new
284 daemon completely replaced the old one.

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

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

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

300 **The move to Wayland**

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

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

311 **The GTK+ and Clutter merger**

312 There has been discussion among GNOME developers recently about merging
313 Clutter and GTK+ into a single toolkit. GTK+ is a powerful toolkit with many
314 years of experience built in, and solving many of the problems posed by complex
315 UIs, but it lacks the eye candy and some of the features people now expect in a
316 modern toolkit. Clutter on the other hand has all of the eye candy and features

317 one expects from a modern toolkit, but lacks the toolkit part. While Mx and St,
318 the GNOME Shell's toolkit, do provide some widgets and higher level features,
319 they are not nearly as fully featured and mature as GTK+. The existence of
320 so many toolkits is being seen as fragmentation of the developer story in the
321 GNOME platform, which also plays a role in these discussions.

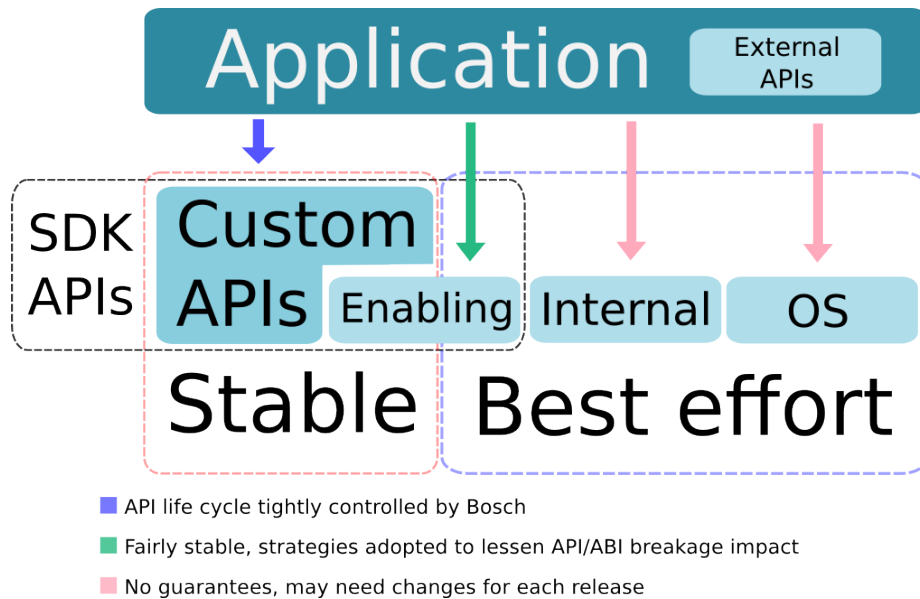
322 When the merger of Clutter and GTK+ happens, the impact and solutions
323 would be pretty much the same as if Clutter and Mx break ABI. Old libraries
324 and applications using Clutter and Mx would remain working, but care would
325 have to be exercised in making sure no process ends up using the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

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

332 **API Support levels**

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

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



346

347 **Custom APIs**

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

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

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

361 **Enabling APIs**

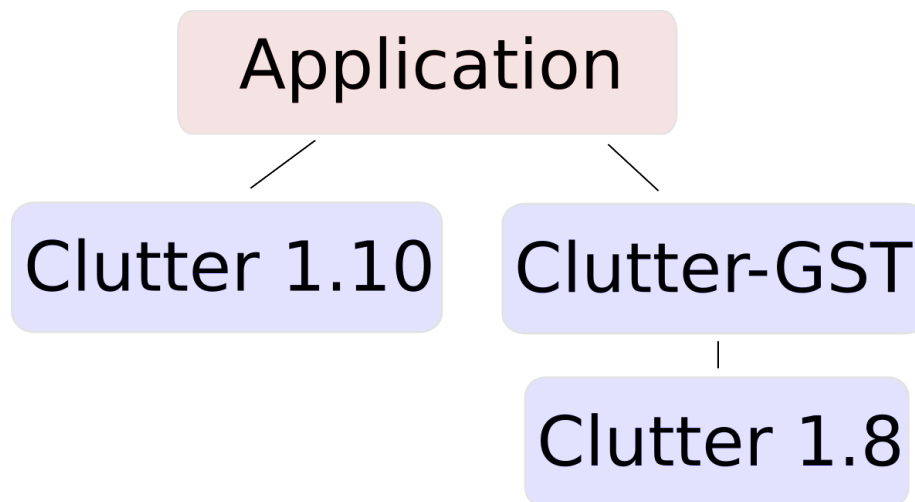
362 These APIs are not guaranteed to be stable between platform upgrades, but
 363 work may be done on a case-by-case basis to provide a smooth migration path,
 364 with old versions coexisting with newer ones when possible. Most existing open
 365 source APIs related to core functionality fall in this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

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

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

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

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



387

388 OS APIs

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

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

393 **Internal APIs**

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

399 **External APIs**

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

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

408 **Differing stability levels**

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

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

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

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

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

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

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

447 **Maintaining API stability**

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

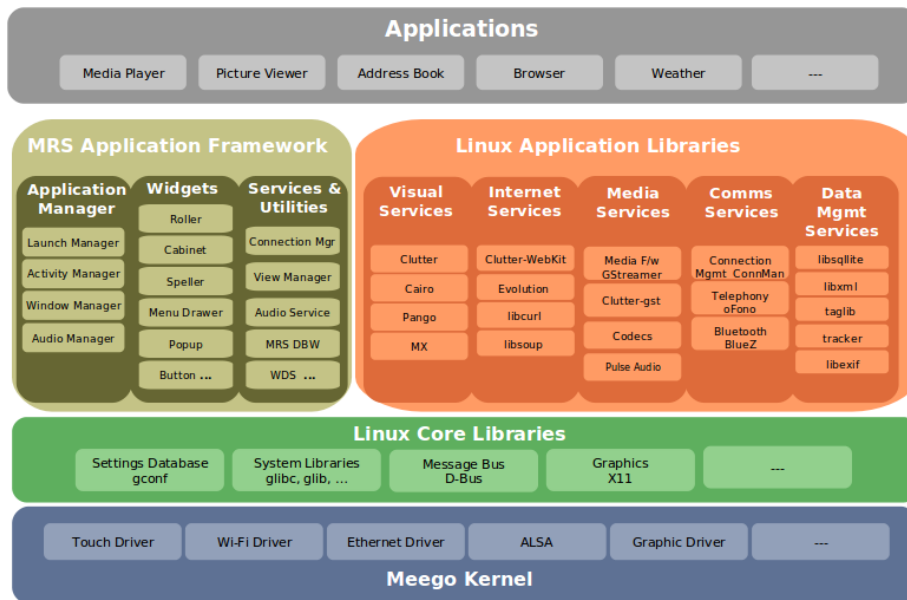
451 The Debian project has some tooling to detect API and ABI changes between
452 releases of a library, though this is invoked at packaging time, which is after the
453 library has been officially released and hence after the damage is done.

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

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

460 **Components**

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



464

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

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

Name	Version	API reference
Telepathy-GLib	0.18	http://telepathy.freedesktop.org/doc/telepathy-glib/
Telepathy-Logger	0.2	http://telepathy.freedesktop.org/doc/telepathy-glib/
Folks	0.6	http://telepathy.freedesktop.org/doc/folks/c/
PulseAudio	1.1	http://freedesktop.org/software/pulseaudio/doxygen/
Bluez	4.98	http://git.kernel.org/?p=bluetooth/bluez.git;a=tree;f=doc
libstartup-notification	0.12	See Notes
libecal	3.3	http://developer.gnome.org/libecal/3.3/
SyncEvolution	1.2	http://api.syncevolution.org/
GUPnP	0.18	http://gupnp.org/docs
libGData	0.11	http://developer.gnome.org/gdata/unstable/
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	http://git.kernel.org/?p=network/ofono/ofono.git;a=tree;f=doc
WebKit-Clutter	1.8.0	
libexif	0.6.20	http://libexif.sourceforge.net/api/
TagLib	1.7	http://developer.kde.org/~wheeler/taglib/api/index.html

471 Conclusion

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

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