



UI customisation

| | | |
|----|--------------------------------------|----------|
| 1 | Contents | |
| 2 | Introduction | 4 |
| 3 | Terminology and Concepts | 4 |
| 4 | Vehicle | 4 |
| 5 | System | 4 |
| 6 | User | 4 |
| 7 | Widget | 4 |
| 8 | User Interface | 4 |
| 9 | Roller | 5 |
| 10 | Speller | 5 |
| 11 | Application Author | 5 |
| 12 | Variant | 5 |
| 13 | View | 5 |
| 14 | Template | 5 |
| 15 | UI prototyping | 5 |
| 16 | WYSIWYG UI editing | 6 |
| 17 | Use Cases | 6 |
| 18 | Multiple Variants | 6 |
| 19 | Fixed Variants | 6 |
| 20 | Templates | 6 |
| 21 | Template Extension | 7 |
| 22 | Custom Widget Usage | 7 |
| 23 | Template Library | 7 |
| 24 | Appearance Customisation | 7 |
| 25 | Different Icon Themes | 7 |
| 26 | Different Fonts | 7 |
| 27 | OTA Updates | 8 |
| 28 | Language | 8 |
| 29 | Right-to-Left Scripts | 8 |
| 30 | OTA Updates | 8 |
| 31 | Animations | 8 |
| 32 | Prototyping | 8 |
| 33 | Day & Night Mode | 9 |
| 34 | View Management | 9 |
| 35 | Display Orientation | 9 |
| 36 | Speed Lock | 9 |
| 37 | Geographical Customisation | 9 |
| 38 | System Enforcement | 9 |
| 39 | Non-Use Cases | 9 |
| 40 | Theming Custom Widgets | 10 |
| 41 | Multiple Monitors | 10 |
| 42 | DPI Independence | 10 |

| | | |
|----|--|-----------|
| 43 | Display Size | 10 |
| 44 | Dynamic Display Resolution Change | 10 |
| 45 | Requirements | 10 |
| 46 | Variant set at Compile-Time | 10 |
| 47 | CSS Styling | 10 |
| 48 | Templates | 11 |
| 49 | Catalogue of Templates | 11 |
| 50 | Template Extension | 12 |
| 51 | Template Modularity | 12 |
| 52 | Custom Widgets in Templates | 12 |
| 53 | Documentation | 12 |
| 54 | Widget Interfaces | 12 |
| 55 | GResources | 13 |
| 56 | MVC Separation | 13 |
| 57 | Language Support | 13 |
| 58 | Animations | 14 |
| 59 | Scripting Support | 14 |
| 60 | Day & Night Mode | 14 |
| 61 | View Management | 14 |
| 62 | Speed Lock | 14 |
| 63 | Scrolling Lists | 14 |
| 64 | Text | 15 |
| 65 | List Columns | 15 |
| 66 | Keyboard | 15 |
| 67 | Pictures | 15 |
| 68 | Video Playback | 15 |
| 69 | Map Gestures | 15 |
| 70 | Web View | 16 |
| 71 | Insensitive Widgets | 16 |
| 72 | Approach | 16 |
| 73 | Templates | 16 |
| 74 | Properties, Signals, and Callbacks | 16 |
| 75 | Widget Factories | 16 |
| 76 | Custom Widgets | 17 |
| 77 | Models | 17 |
| 78 | Theming | 17 |
| 79 | Theme Changes | 18 |
| 80 | Language Support | 18 |
| 81 | Language Changes | 18 |
| 82 | Updating Languages | 18 |
| 83 | Day & Night Mode | 18 |
| 84 | Speed Lock | 18 |
| 85 | List Columns | 19 |
| 86 | Keyboard | 19 |

| | | |
|----|--|----|
| 87 | Insensitive Widgets | 19 |
| 88 | Notifications | 19 |
| 89 | Masking Unknown Applications | 19 |

90 **Introduction**

91 The goal of this user interface customisation design document is to reduce app
92 development time when porting between variants by abstracting the differences
93 between variants into a UI library.

94 The goal of standardising this process is reduce the amount of code written or
95 changed in customising a variant. It is understood that for system components,
96 code might have to be altered for some requests, but code inside application
97 bundles should remain as similar as possible and work in variant-specific ways
98 automatically.

99 **Terminology and Concepts**

100 **Vehicle**

101 For the purposes of this document, a *vehicle* may be a car, car trailer, motor-
102 bike, bus, truck tractor, truck trailer, agricultural tractor, or agricultural trailer,
103 amongst other things.

104 **System**

105 The *system* is the infotainment computer in its entirety in place inside the
106 vehicle.

107 **User**

108 The *user* is the person using the system, be it the driver of the vehicle or a
109 passenger in the vehicle.

110 **Widget**

111 A *widget* is a reusable part of the user interface which can be changed depending
112 on location and function.

113 **User Interface**

114 The *user interface* is the group of all widgets in place in a certain layout to
115 represent a specific use-case.

116 **Roller**

117 The *roller* is a list widget named after a cylinder which revolves around its
118 central horizontal axis. As a result of being a cylinder it has no specific start
119 and finish and appears endless.

120 **Speller**

121 The *speller* is a widget for text input.

122 **Application Author**

123 The *application author* is the developer tasked with writing an application using
124 the widgets described in this document. They cannot modify the variant or the
125 user interface library.

126 **Variant**

127 A *variant* is a customised version of the system by a particular system integrator.
128 Usually variants are personalised with particular colour schemes and logos and
129 potentially different widget behaviour.

130 **View**

131 A *view* is an page in an application with an independent purpose. Views move
132 from one to another, and sometimes also back, to form the workflow of the
133 application. For example, in a photo application the list of photos is one view
134 and the highlight on one photo in particular, perhaps with more metadata from
135 the photo, is another view.

136 **Template**

137 A *template* is a text-based representation of a set of widgets in a view. Templates
138 are for allowing changes and extensions without having to rebuild the actual
139 code.

140 **UI prototyping**

141 *UI prototyping* is the process of building a mock-up of a UI to evaluate how it
142 looks, and how usable it is for different use cases —but without hooking up the
143 UI to an application implementation or backing code. The idea is to be able
144 to produce a representative UI as fast as possible, so designers and testers can
145 evaluate its usability, and can produce further iterations of the design, without
146 wasting time on implementing backing functionality in code until the design is
147 finalised. At this point, a programmer can turn the prototype into a complete
148 implementation in code.

149 The process of prototyping is not relevant to UI *customisation*, but is relevant
150 to the process of using a UI toolkit.

151 Here is an example of some [prototype UIs](#)¹, made in Inkscape.

152 **WYSIWYG UI editing**

153 *WYSIWYG UI editing* is the process of using a UI editor, such as [Glade](#)², where
154 the UI elements can be composed visually and interactively to build the UI, for
155 example by dragging and dropping them together. The appearance of the UI in
156 the designer is almost identical to its appearance when it is run in production.

157 **Use Cases**

158 A variety of use cases for UI customisation are given below.

159 **Multiple Variants**

160 Each system integrator wants to use the same user interface without having to
161 rewrite from scratch (see [Variant differences](#)).

162 For example, in the speller, variant A wants to highlight the key on an on-screen-
163 keyboard such that the key pops out of the keyboard, whereas variant B wants
164 to highlight just the letter within the key with no pop out animation.

165 Another example, in the app launcher, variant A wants to use a cylinder anima-
166 tion for rolling whereas variant B wants to scroll the list of applications like a
167 flat list.

168 **Fixed Variants**

169 A system integrator wants multiple variants to be installable concurrently on
170 the system, but wants the variant in use to be fixed and not able to change
171 after being set in a configuration option. The system integrator wants said
172 configuration option to be changeable without rebuilding.

173 **Templates**

174 A system integrator wants to customise the user interface as easily as possible
175 without recompilation of applications. The system integrator wants to be able
176 to choose the widgets in use in a particular application user interface (from a
177 list of available widgets) and have them work accordingly.

178 For example, in a photo viewing application with one photo selected, system in-
179 tegrator A might want to display the selected photo with nothing else displayed,

¹<https://github.com/gnome-design-team/gnome-mockups/blob/master/passwords-and-keys/passwords-and-keys.png>

²<https://glade.gnome.org/>

180 while system integrator B might want to display the selected photo in the centre
181 of the display, but also have the next and previous photos slightly visible at the
182 sides.

183 **Template Extension**

184 A system integrator wants to use the majority of a provided template, but
185 also wants to add their own variant-specific extensions. The system integrator
186 wants to achieve this without copy and pasting provided templates to retain
187 maintainability, and wants to add their own extension template which merely
188 references the provided one.

189 For example, said system integrator wants to use an provided button widget,
190 but wants to make it spin 360° when clicked. They want to just override the
191 library widget, adding the spin code, and not have to touch any other code
192 relating to the internal working of the widget already provided in the library.

193 **Custom Widget Usage**

194 A system integrator wants to implement custom widgets by writing actual code.
195 The system integrator wants to be integrate the new custom widgets into the
196 user interface and into the developer tooling.

197 **Template Library**

198 A system integrator wants to be able to add new templates to the system via
199 over the air (OTA) updates. The system integrator does not want the template
200 to be able to reload automatically after being updated.

201 **Appearance Customisation**

202 Each system integrator wants to customise the look and feel of applications
203 by changing styling such as padding widths, border widths, colours, logos, and
204 gradients. The system integrator wants to make said modifications with the
205 minimum of modifications, especially to the source code.

206 **Different Icon Themes**

207 Each system integrator wants to be able to trivially change the icon theme in use
208 across the user interface not only without recompilation, but also at runtime.

209 **Different Fonts**

210 Each system integrator wants to be able to trivially change the font in use across
211 the user interface, and bundle new fonts in with variants.

212 **OTA Updates**

213 System integrators want to be able to add fonts using over the air (OTA) updates.
214 For example, the system integrator wants to change the font in use across the
215 user interface of the variant. They send the updated theme definition as well as
216 the new font file via an update and want it to be registered automatically and
217 be immediately useable.

218 **Language**

219 The user wants to change the language of the controls of the system to their
220 preferred language such that every widget in the UI that contains text updates
221 accordingly without having to restart the application.

222 **Right-to-Left Scripts**

223 As above, the user wants to change the language of the controls of the system,
224 but to a language which is read from right-to-left (Arabic, Persian, Hebrew, etc.),
225 instead of left-to-right. The user expects the workflow of the user interface to
226 also change to right-to-left.

227 **OTA Updates**

228 A system integrator wants to be able to add and improve language support over
229 over the air (OTA) updates. For example, the system integrator wants to add
230 a new translation to the system. They send the translation via an update and
231 want the new language to immediately appear as an option for the user to select.

232 **Animations**

233 A system integrator wants to customise animations for the system. For example,
234 they want to be able to change the behaviour of list widgets by setting the
235 visual response using kinetic scrolling and whether there's an elastic effect when
236 reaching the end of items. Another example is they also want to be able to
237 customise the animation used when changing views in an application. Another
238 example is the how button widgets react when pressed.

239 The system integrator then expects to see the changes apply across the entire
240 system.

241 **Prototyping**

242 An application author wants to prototype a UI rapidly (see [UI prototyping](#)),
243 using a WYSIWYG UI development tool (see [WYSIWYG UI editing](#)) with
244 access to all the widgets in the library, including custom and vendor-specific
245 widgets.

246 **Day & Night Mode**

247 A user is using the system when dark outside and wants the colour scheme of
248 the display to change to accommodate for the darkness outside so not be too
249 bright and dazzle the user. Requiring the user to adapt their eyes momentarily
250 for the brightness of the system could be dangerous.

251 **View Management**

252 An application author has several views in their application and doesn't want to
253 have to write a system of managing said views. They want to be able to add a
254 workflow and leave the view construction, show and hide animations, and view
255 destruction up to the user interface library.

256 **Display Orientation**

257 A system integrator changes the orientation of the display. They expect the
258 user interface to adapt and display normally, potentially using a different layout
259 more suited to the orientation.

260 Note that the adaptation is only expected to be implemented if easy and is not
261 expected to be instantaneous, and a restart of the system is acceptable.

262 **Speed Lock**

263 Laws require that when the vehicle is moving some features be disabled or
264 certain behaviour modified.

265 **Geographical Customisation**

266 Different geographical regions have different laws regarding what features and
267 behaviours need to be changed, so it must be customisable (only) by the system
268 integrator when it is decided for which market the vehicle is destined.

269 **System Enforcement**

270 Due to restrictions being government laws, system integrators don't want to rely
271 on application authors to respect said restrictions, and instead want the system
272 to enforce them automatically.

273 **Non-Use Cases**

274 A variety of non-use cases for UI customisation are given below.

275 **Theming Custom Widgets**

276 An application developer wants to write their own widget using a library directly.
277 They understand that standard variant theming will not apply to any custom
278 widget and any integration will have to be achieved manually.

279 Note that although unsupported directly by the user interface library, it is
280 possible for application authors to implement this higher up in the application
281 itself.

282 **Multiple Monitors**

283 A system integrator wants to connect two displays (for example, one via HDMI
284 and one via LVDS) and show something on each one, for example when devel-
285 oping on a target board like the i.MX6. They understand this is not supported
286 by Apertis.

287 **DPI Independence**

288 A system integrator uses a display with a different DPI. They understand that
289 they should not expect that the user interface changes to display normally and
290 not too big/small relative to the old DPI.

291 **Display Size**

292 A system integrator changes the resolution of the display. They understand
293 that they should not expect the user interface to adapt and display normally,
294 potentially using a different layout more suited to the new display size.

295 **Dynamic Display Resolution Change**

296 A system integrator wants to be able to change the resolution of the display or
297 resize the user interface. They understand that a dynamic change in the user
298 interface is not supported in Apertis.

299 **Requirements**

300 **Variant set at Compile-Time**

301 Multiple variants should be supported on the system but the variant in use
302 should be decided at application compile-time such that it cannot be changed
303 later (see **Fixed variants**).

304 **CSS Styling**

305 The basic appearance of the widgets should be stylable using CSS, changing
306 the look and feel as much as possible with no modifications to the source code

307 required (see [Appearance customisation](#), [Different icon themes](#)).

308 The changes possible using CSS do not need to be incredibly intrusive and are
309 limited to the basic core CSS properties. For example, changing colour scheme
310 (background-color, color), icon theme & logos (background-image), fonts (font-
311 family, font-size), and spacing (margin, padding).

312 More intrusive changes to the user interface should be achieved using templates
313 (see [Templates](#)) instead of CSS changes.

314 For example, a system integrator wants to change the colour of text in buttons.
315 This should be possible by changing some CSS.

316 **Templates**

317 CSS is appropriate for changing simple visual aspects of the user interface but
318 does not extend to allow for structural modifications to applications (see [CSS](#)
319 [styling](#)). Repositioning widgets or even changing which widgets are to be used is
320 not possible with CSS and should be achieved using templates (see [Templates](#)).

321 There are multiple layers of widgets available for use in applications. Starting
322 from the lowest, simplest, level and moving higher, encapsulating more with
323 each step:

- 324 • buttons, entries, labels, ...
- 325 • buttons with labels, radio buttons with labels, ...
- 326 • lists, tree view, ...
- 327 • complete views, or *templates*.

328 Templates are declarative representations of the layout of the user interface
329 which are read at runtime by the application. Using templates it is possible
330 to redesign the layout, look & feel, and controls of the application without
331 recompilation.

332 The purpose of templates is to reduce the effort required by an application
333 author to configure each widget, and to maintain the same look and feel across
334 the system.

335 **Catalogue of Templates**

336 There should be a catalogue of templates provided by the library which system
337 integrators can use to design their applications (see [Template library](#)). The
338 layouts of applications should be limited to the main use cases.

339 For example, one system integrator could want the music application to be
340 a simple list of albums to choose from, while another could want the same
341 information represented in a grid. This simple difference should be possible by
342 using different templates already provided by the user interface library.

343 **Template Extension**

344 In addition to picking layouts from user interface library-provided templates, it
345 should also be possible to take existing templates and change them with the
346 minimal of copy & pasting (see [Template extension](#)).

347 For example, a system integrator could want to change the order of labels in
348 a track information view. The default order in the library-provided template
349 could be track name and then artist name, but said system integrator wants
350 the artist name first, followed by the track name. This kind of change is too
351 fundamental to do in CSS so a template modification is required. The system
352 integrator should be able to take the existing library-provided template and
353 make minimal modifications and minimal copy & pasting to change the order.

354 **Template Modularity**

355 Templates should be as modular as possible in order to break up the parts of
356 a design into smaller parts. This is useful for when changes are required by a
357 system integrator (see [Templates](#), [Template extension](#)). If the entire layout is
358 in one template, it is difficult to make small changes without having to copy the
359 entire original template.

360 Fine-grained modularity which leads to less copy & pasting is optimal because
361 it makes the template more maintainable, as there's only one place to change if
362 a bug is discovered in the original library-provided template.

363 **Custom Widgets in Templates**

364 A system integrator should be able to use custom widgets they have written
365 for the particular variant in the template format (see [Custom widget usage](#)).
366 The responsibility of compatibility with the rest of the user interface of custom
367 widgets is on the widget author.

368 **Documentation**

369 With a library of widgets and models available to the system integrator, the
370 options of widgets and ways to interact with them should be well documented
371 (see [Template library](#)). If signals, signal callbacks, and properties are provided
372 these should all be listed in the documentation for the system integrator to
373 connect to properly.

374 **Widget Interfaces**

375 When swapping a widget out for another one in a template it is important that
376 the API matches so the change will work seamlessly. To ensure this, widgets
377 should implement core interfaces (button, entry, combobox, etc.) so that when
378 swapped out, views will continue to work as expected using the replacement
379 widget. Applications should only use API which is defined on the interface, not

380 on the widget implementation, if they wish for their widgets to be swappable
381 for those in another variant.

382 As a result, system integrators swapping widgets out for replacements should
383 check the API documentation to ensure that the interface implemented by the
384 old widget is also implemented in the new widget. This will ensure compatibility.

385 **GResources**

386 If an application is loading a lot of templates from disk there could be an
387 overhead in the input/output operation in loading them. A way around this
388 is to use [GResource](#)³s. GResources are useful for storing arbitrary data, such
389 as templates, either packed together in one file, or inside the binary as literal
390 strings. It should be noted that if linked into the binary itself, the binary will
391 have to be rebuilt every time the template changes. If this is not an option,
392 saving the templates in an external file using the `glib-compile-resources` binary
393 is necessary.

394 The advantage of linking resources into the binary is that once the binary is
395 loaded from disk there is no more disk access. The disadvantage of this is as
396 mentioned before is that rebuilding is required every time resources change. The
397 advantage of putting resources into a single file is that they are only required to
398 be mapped in memory once and then can be shared among other applications.

399 **MVC Separation**

400 There should be a functional separation between data provider (*model*), the
401 way in which it is displayed in the user interface (*view*), and the widgets for
402 interaction and data manipulation (*controller*) (see example in [Templates](#)). The
403 model should be a separate object not depending on any visual aspect of the
404 widget.

405 Following on from the previous example (in [Templates](#)), the model would be the
406 list of pictures on the system, and the two variants would use different widgets,
407 but would attach the same model to each widget. This is the key behind being
408 able to swap one widget for another without making code changes.

409 This separation would push the *model* and *controller* responsibility to the user
410 interface library, and an application would only depend on the *model* in that it
411 provides the data to fill said model.

412 **Language Support**

413 All widgets should be linked into a language translation system such that it is
414 trivial not only for the user to change language (see [Language](#)), but also for new
415 translations to be added and existing translations updated (see [Ota updates](#)).

³<https://developer.gnome.org/gio/stable/GResource.html>

416 Animations

417 Animations in use in widgets should be configurable by the system integrator
418 (see [Animations](#) for examples). These animations should be used widely across
419 the system to ensure a consistent experience. Applications should expose a fixed
420 set of transitions which can be animated so system integrators can tell what can
421 be customised.

422 Scripting Support

423 The widgets and templates should be usable from a UI design format, such
424 as [GtkBuilder](#)⁴. This includes custom widgets. This would enable application
425 authors to quickly prototype applications (see [Prototyping](#)).

426 Day & Night Mode

427 The user interface should change between light and dark mode when outside the
428 vehicle becomes dark in order to not shine too brightly and distract the user
429 (see [Day night mode](#)).

430 View Management

431 A method of managing application views (see [View](#)) should be provided to ap-
432 plication authors (see [View management](#)). On startup the application should
433 provide its views to the view manager. From this point on the responsibility
434 of constructing views, switching views, and showing view animations should be
435 that of the view manager. The view manager should pre-empt the construction
436 of views, but also be sensitive to memory usage so not load all views simultane-
437 ously.

438 Speed Lock

439 Some features and certain behaviour in the user interface should be disabled or
440 modified respectively when the vehicle is moving (see [Speed lock](#)). It should be
441 possible to customise whether each item listed below is disabled or not as it can
442 depend on the target market of the vehicle (see [Geographical customisation](#)).
443 Additionally, it should be up to the system to enforce the disabling of the fol-
444 lowing features and should not be left completely up to application authors (see
445 [System enforcement](#)).

446 Scrolling Lists

447 The behaviour of gestures in scrolling lists should be altered to remove fast move-
448 ments with many screen updates. Although still retaining similar functionality,
449 gestures should cause far fewer visual changes. For example, swiping up would
450 no longer start a kinetic scroll, but would move the page up one tabulation.

⁴<https://developer.gnome.org/gtk3/stable/GtkBuilder.html#GtkBuilder.description>

451 **Text**

452 Text displayed should either be masked or altered to remove the distraction of
453 reading it while operating the vehicle, depending on the nature of the text.

- 454 • SMS messages and emails can have dynamic content so they should be
455 hidden or masked.
- 456 • Help text or dialog messages should have alternate, shorter messages to
457 be shown when the speed lock is active.

458 **List Columns**

459 Lists with columns should limit the number of columns visible to ensure super-
460 fluous information is not distracting. For example, in a contact list, instead of
461 showing both name and telephone number, the list could show only the
462 name.

463 **Keyboard**

464 The keyboard should be visibly disabled and not usable.

465 Additionally, default values should be available so that operations can succeed
466 without the use of a keyboard. For example when adding a bookmark when
467 the vehicle is stationary the user will be able to choose a name for the new
468 bookmark before saving it. When the vehicle is moving the bookmark will be
469 automatically saved under a default name without the user being prompted for
470 the name. The name (and other use cases of default values) should be modifiable
471 later.

472 **Pictures**

473 Superfluous pictures used in applications as visual aids which could be distract-
474 ing should be hidden. For example, in the music application, album covers
475 should be hidden from the user.

476 **Video Playback**

477 Video playback must either be paused or the video masked (while the audio
478 continues to sound).

479 **Map Gestures**

480 As with kinetic scrolling in lists (see [Scrolling lists](#)), the gestures in the map
481 widget should make fewer visual changes and reduce the number of distractions
482 for the user. Similar to the kinetic scroll example, the map view should move
483 by a fixed distance instead of following the user's input.

484 **Web View**

485 Any web view should be masked and not showing any content.

486 **Insensitive Widgets**

487 When aforementioned functionality is disabled by the speed lock, it should be
488 made clear to the user what has been modified and why.

489 **Approach**

490 **Templates**

491 The goal of templates is to allow an application developer to change the user
492 interface of their application without having to changing the source code. These
493 are merely templates and have no way of implementing logic (if/else statements).
494 If this is required, widget code customisation is required (see [Custom widgets](#)).

495 **Properties, Signals, and Callbacks**

496 The GObject properties that can be set, the signals that can be connected
497 to, and the signal callbacks that can be used, should be listed clearly in the
498 application documentation. This way, system integrators can customise the
499 look and feel of the application using already-written tools.

500 When changing a template to use a different widget it might be necessary to
501 change the signal callbacks. This largely depends on the nature of the change
502 of widget but signals names and signatures should be as consistent as possible
503 across widgets to enable changing as easily as possible. If custom callbacks
504 are used in the code of an application, and the callback signature changes,
505 recompilation will be necessary. The signals emitted by widgets and their type
506 signatures are defined in their interfaces, documented in the API documentation.

507 **Widget Factories**

508 If a system integrator wants to replace a widget everywhere across the user
509 interface, they can use a widget factory to replace all instances of said old
510 widget with the new customised one.

511 For example, if a system integrator wants to stop using `LightwoodButtons` and
512 instead use the custom `FancyButton` class, there are no changes required to any
513 template, but an entry is added to the widget factory to produce a `FancyButton`
514 whenever a `LightwoodButton` is requested. Templates can continue referring to
515 `LightwoodButton` or can explicitly request a `FancyButton` but both will be created
516 as `FancyButtons`. If an application truly needs the older `LightwoodButton`, it needs
517 to create a subclass of `LightwoodButton` which is not overridden by anything, and
518 then refer to that explicitly in the template.

519 Custom Widgets

520 Widgets can be subclassed by system integrators in variants and used by ap-
521 plication developers by creating shared libraries linking to the widget library.
522 Applications then link to said new library and once the new widgets are reg-
523 istered with the GObject type system they can be referred to in ClutterScript
524 user interface files. If a system integrator wants a radically different widget,
525 they can write something from scratch, ensuring to implement the appropriate
526 interface. Subclassing existing widgets is for convenience but not technically
527 necessary.

528 Widgets should be as modularised as possible, splitting functionality into virtual
529 methods where a system integrator might want to override it. For example,
530 if a system integrator wants the roller widget to have a different activation
531 animation depending on the number of items in the model, they could create
532 a roller widget subclass, and override the appropriate virtual methods (in this
533 case activate) and update the animation as appropriate:

534 Models

535 Data that is to be displayed to the user in list widgets should be stored in an
536 orthogonal model object. This object should have no dependency on anything
537 visual (see [MVC separation](#)).

538 The actual implementation of the model should be of no importance to the
539 widgets, and only basic model interface methods should be called by any widget.
540 It is suggested to use the [GListModel](#)⁵ interface as said model interface as it
541 provides a set of simple type-safe methods to enumerate, manipulate, and be
542 notified of changes to the model.

543 As `GListModel` is only an interface, an implementation of said interface should
544 be written, ensuring to implement all methods and signals, like `GListStore`.

545 Theming

546 Using the `GtkStyleContext` object from GTK+ is wise for styling widgets as it
547 can aggregate styling information from many sources, including CSS. GTK+'
548 s CSS parsing code is advanced and well tested as GTK+ itself switched its
549 [Adwaita](#)⁶ default theme to pure CSS some time ago, replacing theme engines
550 that required C code to be written to customise appearance.

551 Said parser and aggregator support multiple layers of overrides. This means
552 that CSS rules can be given priorities and rules are followed in a specific order
553 (for example theme rules are set, and can be overridden by variant rules, and can
554 be overridden by application rules, where necessary). This is ideal for Apertis
555 where themes set defaults and variants need only make changes where necessary.

⁵<https://developer.gnome.org/gio/stable/GListModel.html>

⁶<https://git.gnome.org/browse/gtk+/tree/gtk/theme/Adwaita>

556 Theme Changes

557 Applications should listen to a documented [GSettings](#)⁷ key for changes to the
558 theme and icon theme. Changes to the theme should update the style properties
559 in the `GtkStyleContext` and will trigger a widget redraw and changes to the icon
560 theme should update the icon paths and trigger icon redraws.

561 Language Support

562 GNU [gettext](#)⁸ is a well-known system for managing translations of applications.
563 It provides tools to scan source code looking for translatable strings and a library
564 to resolve said strings against language files which are easily updated without
565 touching the source code of said applications.

566 Language Changes

567 Applications should listen to a documented GSettings key for changes to the
568 user-chosen language, then re-translate all strings and redraw.

569 Updating Languages

570 Language files for GNU [gettext](#) saved into the appropriate directory can be
571 easily used immediately with no other changes to the application. Over the
572 air (OTA) updates can contain updated language files which get saved to the
573 correct location and would be loaded the next time the application is started.

574 Day & Night Mode

575 Inspired by GTK+'s [dark mode](#)⁹, variant CSS should provide a `dark` class for
576 widgets to be used in night mode. If the `dark` class is not set the user interface
577 should be in day mode. [CSS transitions](#)¹⁰ should make the animation smooth.

578 A central GSettings key should be read to know when the system is in day or
579 night mode. It will be modifiable for testing and in development.

580 Speed Lock

581 There should be a system-operated service that determines when the vehicle
582 is moving and when it is stationary. From this point the Apertis widgets and
583 applications should change when and where appropriate.

584 There should be a GSettings key which indicates whether the speed lock is active
585 or not. This key should only be modifiable by said system-operated service and
586 should be readable by the entire system.

⁷<https://developer.gnome.org/gio/stable/GSettings.html>

⁸<https://www.gnu.org/software/gettext/>

⁹<https://developer.gnome.org/gtk3/stable/GtkSettings.html#GtkSettings--gtk-application-prefer-dark-theme>

¹⁰http://www.w3schools.com/css/css3_animations.asp

587 **List Columns**

588 The number of columns visible should be reduced to remove superfluous infor-
589 mation when the speed lock is active (see **List columns**). The nature of every
590 list can be different and the detection of superfluous information is impossible
591 automatically. There should be a way of either application authors specifying
592 which columns should be hidden, or it should be left up to the application itself.
593 If the latter is not an option (see enforcement comments in **Speed lock**), the
594 entire list widget should be masked to hide its contents.

595 **Keyboard**

596 As mentioned in **Keyboard**, applications should deal with the possibility that
597 the keyboard may not be available at any given time, if the speed lock is active.
598 In the case that the keyboard request is denied, the application should change
599 its user experience slightly to accommodate for this, such as the example with
600 bookmarks given previously.

601 The change of user experience also means there must be other ways in which
602 users can edit named items using default values after the speed lock has been
603 disabled.

604 **Insensitive Widgets**

605 As highlighted in **Insensitive widgets**, it should be made obvious to the user when
606 functionality is disabled, and why. There should be a uniform visual change
607 to widgets when they have been made insensitive so users can immediately
608 recognise what is happening.

609 A documented CSS class should be added to widgets that are made insensitive
610 by the speed lock so that said widgets follow an identical change in display.

611 **Notifications**

612 Pop-up notifications or a status bar message should make it clear to the user that
613 the speed lock is active and if appropriate, highlight the current functionality
614 that has been disabled.

615 **Masking Unknown Applications**

616 Applications can technically implement custom widgets and not respect the
617 rules of the speed lock. As a result, applications which haven't been vetted
618 by an approved authority should not be able to be run when the speed lock is
619 active. When they are already running and the speed lock is activated, they
620 should be masked and the user should not be able to interact with them.

621 This behaviour should be customisable and possibly only enabled in a region in
622 which laws are very strict about speed lock restrictions.