



UI customisation

# 1 Contents

2	<b>UI customisation</b>	<b>2</b>
3	Introduction . . . . .	2
4	Terminology and Concepts . . . . .	2
5	Vehicle . . . . .	2
6	System . . . . .	2
7	User . . . . .	3
8	Widget . . . . .	3
9	User Interface . . . . .	3
10	Roller . . . . .	3
11	Speller . . . . .	3
12	Application Author . . . . .	3
13	Variant . . . . .	3
14	View . . . . .	3
15	Template . . . . .	4
16	UI prototyping . . . . .	4
17	WYSIWYG UI editing . . . . .	4
18	Use Cases . . . . .	4
19	Multiple Variants . . . . .	4
20	Templates . . . . .	5
21	Appearance Customisation . . . . .	6
22	Different Icon Themes . . . . .	6
23	Different Fonts . . . . .	6
24	Language . . . . .	6
25	Animations . . . . .	7
26	Prototyping . . . . .	7
27	Day & Night Mode . . . . .	7
28	View Management . . . . .	7
29	Display Orientation . . . . .	7
30	Speed Lock . . . . .	7
31	Non-Use Cases . . . . .	8
32	Theming Custom Clutter Widgets . . . . .	8
33	Multiple Monitors . . . . .	8
34	DPI Independence . . . . .	8
35	Display Size . . . . .	8
36	Dynamic Display Resolution Change . . . . .	9
37	Requirements . . . . .	9
38	<b>Variant set at Compile-Time</b> . . . . .	9
39	<b>CSS Styling</b> . . . . .	9
40	Templates . . . . .	9
41	<b>MVC Separation</b> . . . . .	11
42	Language Support . . . . .	12
43	Animations . . . . .	12
44	Scripting Support . . . . .	12
45	Day & Night Mode . . . . .	12

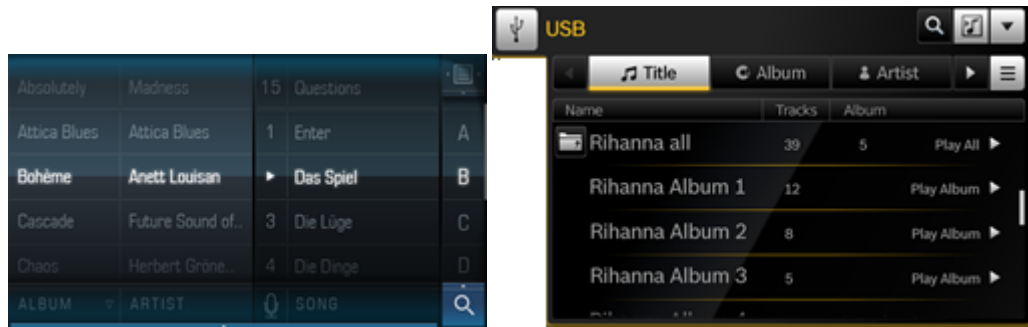
46	View Management . . . . .	12
47	Speed Lock . . . . .	13
48	Approach . . . . .	14
49	Templates . . . . .	14
50	Models . . . . .	21
51	Theming . . . . .	21
52	Language Support . . . . .	22
53	Day & Night Mode . . . . .	23
54	View Management . . . . .	23
55	Speed Lock . . . . .	23
56	References . . . . .	25
57	GTK+ Migration . . . . .	25
58	Appendix . . . . .	28
59	Variant Differences . . . . .	28

## 60 UI customisation

### 61 Introduction

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

65 For example, below are designs of an audio player application — on the left is  
 66 variant A and on the right is the variant B.



68 The A variant mixes three independently scrollable lists for artist, album, and  
 69 then track. The B variant uses one scrollable list with columns for the afore-  
 70 mentioned details. Using different widgets and styling it should be possible to  
 71 radically change the user interface as above. More examples of variant changes  
 72 are shown in [Variant differences](#).

73 The goal of standardising this process is reduce the amount of code written or  
 74 changed in customising a variant. It is understood that for system components,  
 75 code might have to be altered for some requests, but code inside application

76 bundles should remain as similar as possible and work in variant-specific ways  
77 automatically.

## 78 **Terminology and Concepts**

### 79 **Vehicle**

80 For the purposes of this document, a *vehicle* may be a car, car trailer, motorbike,  
81 bus, truck tractor, truck trailer, agricultural tractor, or agricultural trailer,  
82 amongst other things.

### 83 **System**

84 The *system* is the infotainment computer in its entirety in place inside the  
85 vehicle.

### 86 **User**

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

### 89 **Widget**

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

### 92 **User Interface**

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

### 95 **Roller**

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

### 99 **Speller**

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

### 101 **Application Author**

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

105 **Variant**

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

109 **View**

110 A *view* is an page in an application with an independent purpose. Views move  
111 from one to another, and sometimes also back, to form the workflow of the  
112 application. For example, in a photo application the list of photos is one view  
113 and the highlight on one photo in particular, perhaps with more metadata from  
114 the photo, is another view.

115 **Template**

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

119 **UI prototyping**

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

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

130 Here is an example of some [prototype UIs](#)<sup>1</sup>, made in Inkscape.

131 **WYSIWYG UI editing**

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

136 This could be contrasted with designing a UI by writing a [ClutterScript](#)<sup>3</sup> file, for

---

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

<sup>2</sup><https://glade.gnome.org/>

<sup>3</sup><https://developer.gnome.org/clutter/stable/ClutterScript.html#ClutterScript.description>

137 example, where the UI has to be run as part of a program in order to visualise  
138 it.

## 139 Use Cases

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

### 141 Multiple Variants

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

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

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

### 150 Fixed Variants

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

### 155 Templates

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

160 For example, in a photo viewing application with one photo selected, system  
161 integrator A might want to display the selected photo with nothing else dis-  
162 played, while system integrator B might want to display the selected photo in  
163 the centre of the display, but also have the next and previous photos slightly  
164 visible at the sides.

### 165 Template Extension

166 A system integrator wants to use the majority of an Apertis-provided template,  
167 but also wants to add their own variant-specific extensions. The system integra-  
168 tor wants to achieve this without copy and pasting Apertis-provided templates  
169 to retain maintainability, and wants to add their own extension template which  
170 merely references the Apertis-provided one.

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

### 175 **Custom Widget Usage**

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

### 179 **Template Library**

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

### 183 **Appearance Customisation**

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

### 188 **Different Icon Themes**

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

### 191 **Different Fonts**

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

### 194 **OTA Updates**

195 System integrators want to be able to add fonts using over the air (OTA) up-  
196 dates. For example, the system integrator wants to change the font in use across  
197 the user interface of the variant. They send the updated theme definition as  
198 well as the new font file via an update and want it to be registered automatically  
199 and be immediately useable.

### 200 **Language**

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

204 **Right-to-Left Scripts**

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

209 **OTA Updates**

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

214 **Animations**

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

221 The system integrator then expects to see the changes apply across the entire  
222 system.

223 **Prototyping**

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

228 **Day & Night Mode**

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

233 **View Management**

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



238 **Display Orientation**

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

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

244 **Speed Lock**

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

247 **Geographical Customisation**

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

251 **System Enforcement**

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

255 **Non-Use Cases**

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

257 **Theming Custom Clutter Widgets**

258 An application developer wants to write their own widget using the Clutter  
259 library directly. They understand that standard variant theming will not apply  
260 to any custom widget and any integration will have to be achieved manually.

261 Note that although unsupported directly by the Apertis user interface library, it  
262 is possible for application authors to implement this higher up in the application  
263 itself.

264 **Multiple Monitors**

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

## 269 **DPI Independence**

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

## 273 **Display Size**

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

## 277 **Dynamic Display Resolution Change**

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

## 281 **Requirements**

### 282 **Variant set at Compile-Time**

283 Multiple variants should be supported on the system but the variant in use  
284 should be decided at application compile-time such that it cannot be changed  
285 later (see [Fixed variants](#)).

### 286 **CSS Styling**

287 The basic appearance of the widgets should be stylable using CSS, changing  
288 the look and feel as much as possible with no modifications to the source code  
289 required (see [Appearance customisation](#), [Different icon themes](#)).

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

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

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

### 298 **Templates**

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

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

- 306 • buttons, entries, labels, ...
- 307 • buttons with labels, radio buttons with labels, ...
- 308 • lists, tree view, ...
- 309 • complete views, or *templates*.

310 Templates are declarative representations of the layout of the user interface  
311 which are read at runtime by the application. Using templates it is possible  
312 to redesign the layout, look & feel, and controls of the application without  
313 recompilation.

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

### 317 **Catalogue of Templates**

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

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

### 325 **Template Extension**

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

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

### 336 **Template Modularity**

337 Templates should be as modular as possible in order to break up the parts of  
338 a design into smaller parts. This is useful for when changes are required by a  
339 system integrator (see [Templates](#), [Template extension](#)). If the entire layout is

340 in one template, it is difficult to make small changes without having to copy the  
341 entire original template.

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

### 345 **Custom Widgets in Templates**

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

### 350 **Documentation**

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

### 356 **Widget Interfaces**

357 When swapping a widget out for another one in a template it is important that  
358 the API matches so the change will work seamlessly. To ensure this, widgets  
359 should implement core interfaces (button, entry, combobox, etc.) so that when  
360 swapped out, views will continue to work as expected using the replacement  
361 widget. Applications should only use API which is defined on the interface, not  
362 on the widget implementation, if they wish for their widgets to be swappable  
363 for those in another variant.

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

### 367 **GResources**

368 If an application is loading a lot of templates from disk there could be an  
369 overhead in the input/output operation in loading them. A way around this  
370 is to use [GResource](#)<sup>4</sup>s. GResources are useful for storing arbitrary data, such  
371 as templates, either packed together in one file, or inside the binary as literal  
372 strings. It should be noted that if linked into the binary itself, the binary will  
373 have to be rebuilt every time the template changes. If this is not an option,  
374 saving the templates in an external file using the `glib-compile-resources` binary  
375 is necessary.

---

<sup>4</sup><https://developer.gnome.org/gio/stable/GResource.html>

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

### 381 **MVC Separation**

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

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

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

### 394 **Language Support**

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

### 398 **Animations**

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

### 404 **Scripting Support**

405 The widgets and templates should be usable from a UI design format, such  
406 as [GtkBuilder](#)<sup>5</sup> or [ClutterScript](#)<sup>6</sup>. This includes custom widgets. This would  
407 enable application authors to quickly prototype applications (see [Prototyping](#)).

---

<sup>5</sup><https://developer.gnome.org/gtk3/stable/GtkBuilder.html#GtkBuilder.description>

<sup>6</sup><https://developer.gnome.org/clutter/stable/ClutterScript.html#ClutterScript.description>

## 408 **Day & Night Mode**

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

## 412 **View Management**

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

## 420 **Speed Lock**

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

## 428 **Scrolling Lists**

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

## 433 **Text**

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

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

## 440 **List Columns**

441 Lists with columns should limit the number of columns visible to ensure super-  
442 fluous information is not distracting. For example, in a contact list, instead of

443 showing both name and telephone number, the list could should show only the  
444 name.

#### 445 **Keyboard**

446 The keyboard should be visibly disabled and not usable.

447 Additionally, default values should be available so that operations can succeed  
448 without the use of a keyboard. For example when adding a bookmark when  
449 the vehicle is stationary the user will be able to choose a name for the new  
450 bookmark before saving it. When the vehicle is moving the bookmark will be  
451 automatically saved under a default name without the user being prompted for  
452 the name. The name (and other use cases of default values) should be modifiable  
453 later.

#### 454 **Pictures**

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

#### 458 **Video Playback**

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

#### 461 **Map Gestures**

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

#### 466 **Web View**

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

#### 468 **Insensitive Widgets**

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

#### 471 **Approach**

#### 472 **Templates**

473 The goal of templates is to allow an application developer to change the user  
474 interface of their application without having to changing the source code. These

475 are merely templates and have no way of implementing logic (if/else statements).  
476 If this is required, widget code customisation is required (see [Custom widgets](#)).

### 477 **Specification in ClutterScript**

478 ClutterScript is a method for creating user interfaces from JSON files. An  
479 example is shown below which describes variant A application chooser user  
480 interface:

```
481 [{
482   "id": "model-categories",
483   "type": "LightwoodAppCategoryModel"
484 },
485 {
486   "id": "model-apps",
487   "type": "LightwoodAppModel"
488 },
489 {
490   {
491     "id": "window",
492     "type": "LightwoodWindow",
493     "children": [
494       {
495         "id": "roller-categories",
496         "type": "LightwoodRoller",
497         "model": "model-categories",
498         "app-list": "roller-apps",
499         "signals": [
500           { "name": "activated", "handler": "category_activated_cb" }
501         ]
502       },
503       {
504         "id": "roller-apps",
505         "type": "LightwoodRoller",
506         "model": "model-apps",
507         "signals": [
508           { "name": "activated", "handler": "app_activated_cb" }
509         ]
510       }
511     ]
512   }
}]
```

513 The first two objects created (`model-categories` and `model-apps`) are models for  
514 the application categories available on the system, and the applications avail-  
515 able on the system—due to their class names (`LightwoodAppCategoryModel` and  
516 `LightwoodAppModel` respectively). These models are not widgets visible in the  
517 user interface, but proper widgets will refer to them later in the template.



518 The next entry describes the main window in the user interface, inside of which  
519 there is a horizontal box (with some style properties set), with two children that  
520 are both of type `LightwoodRoller`. Although these widgets are of the same type,  
521 they are different instances and they have been given different models. The first  
522 roller widget (on the left) has been given the `model-categories` model and the  
523 second roller widget (on the right) has been given the `model-apps` model, both  
524 created at the beginning of the JSON file.

525 Additionally the `LightwoodRoller::activated` signal is connected on both rollers  
526 to different callbacks. The signal callback names are listed in the application  
527 documentation. In this case, when the left-hand roller with categories is changed  
528 (activated), the right-hand roller with applications is updated (set by the `app-`  
529 `list` property).

530 Another example to compare is given below with the B-variant application  
531 chooser user interface:

```
532 [{  
533   "id": "model-apps",  
534   "type": "LightwoodAppModel"  
535 },  
536  
537 {  
538   "id": "window",  
539   "type": "LightwoodWindow",  
540   "children": [  
541     {  
542       "id": "list-apps",  
543       "type": "LightwoodList",  
544       "model": "model-apps",  
545       "signals": [  
546         { "name": "activated", "handler": "app_activated_cb" }  
547       ]  
548     }  
549   ]  
550 }]
```

551 The differences of the B-variant application chooser in comparison to the A-  
552 variant application chooser are:

- 553 1. There is no categories model and no categories roller.
- 554 2. There is no more box inside the main window widget.
- 555 3. The list widget is a `LightwoodList` instead of a `LightwoodRoller`. This is  
556 a visual difference dictated by the widget implementation and chosen for  
557 this variant, but the data backend for both lists (in the model `model-apps`)  
558 is unchanged. Both widgets should implement a common `LightwoodCol-`  
559 `lection` interface.

560 These are just two examples of how an application chooser could be designed.  
561 The user interface files contain minimal theming as that is achieved in separate  
562 CSS files (see [Theming](#)).

563 Typically, applications will come with many templates for system integrators to  
564 either use, or take guidance from.

### 565 **Properties, Signals, and Callbacks**

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

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

577 For example, in the examples above, a `LightwoodRoller` widget for listing ap-  
578 plications was changed to a `LightwoodList` and the activated signal remained  
579 connected to the same `app_activated_cb` callback.

### 580 **Template Inheritance**

581 At the time of writing, `ClutterScript` has no way of referring to objects from other  
582 JSON files or of making an object a modified version of another. A proposal  
583 to modify `ClutterScriptParser` to support this feature is as follows (this change  
584 would take a couple of days to implement):

585 An `app-switcher.json` file contains the following objects defined:

```
586 [{  
587   "id": "view-header",  
588   "type": "LightwoodHeader",  
589   "height": 100,  
590   "width": 200  
591 },  
592  
593 {  
594   "id": "view-footer",  
595   "type": "LightwoodFooter",  
596   "height": 80  
597 },  
598  
599 {
```

```

600   "id": "app-switcher",
601   "type": "LightwoodWindow",
602   "color": "#ff0000",
603   "children": [ "view-header", ... , "view-footer" ]
604 }]

```

Header and footer objects are defined (`view-header` and `view-footer`) each with height properties (100 and 80 respectively). An `app-switcher` object is also created with the `color` and `children` properties set. Note that the `children` property is a list referring to the header and footer objects created before. (The ellipsis between said objects marks the omission of other objects between header and footer for brevity.)

If a system integrator wanted to give the same appearance to their app switcher view but wanted to change the height of the header and the colour of the main app switcher, without copy & pasting a lot of text to redefine all these objects, objects can simply extend on previous definitions. For example, in a `my-app-switcher.json`:

```

616 [{
617   "external-uri": "file:///path/to/app-switcher.json",
618   "id": "view-header",
619   "height": 120
620 },
621
622 {
623   "external-uri": "file:///path/to/app-switcher.json",
624   "id": "app-switcher",
625   "color": "#ffff00"
626 }]

```

Referencing objects defined in other files can be achieved by specifying the `external-uri` property pointing to the other file, and the `id` property for selecting the external object.

In this example, the `view-header` object is extended and the `height` property is set to 120. All other properties on the original object remain untouched. For example, the `width` property of the header remains at 200.

It is possible to simply to refer to objects in other files without any changes. Each external object must be referred to separately as they are not automatically brought into scope after the first `external-uri` reference. For example:

```

636 {
637   "id": "example-with-children",
638   ...
639   "children": [
640     "first-child",
641     {

```

```

642     "id": "second-child",
643     "type": "ExampleType"
644   },
645   {
646     "external-uri": "file:///path/to/another.json",
647     "id": "third-child"
648   }
649 ]
650 }

```

651 In this example, this object has three children:

- 652 1. An object called `first-child`, defined elsewhere in the JSON file.
- 653 2. An object called `second-child`, defined inline of type `ExampleType`.
- 654 3. An object called `third-child`, defined in `another.json`.

655 In these examples, the `external-uri` used the `file://` URI scheme, but others  
656 supported by GIO can be used. For example, templates in [GResources](#) can be  
657 used using the `resource://` URI scheme.

658 Application authors should not use templates and inheritance excessively such  
659 that every single object is in a separate file. This will cause more disk activity  
660 and could potentially slow down the application. Templates should be broken  
661 up when clarity is in question or when a non-trivial object is to be used across  
662 in other views.

## 663 **Widget Factories**

664 If a system integrator wants to replace a widget everywhere across the user  
665 interface, they can use a widget factory to replace all instances of said old  
666 widget with the new customised one. This is achieved by using a factory which  
667 overrides the type parameter in a `ClutterScript` object.

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

## 676 **Custom Widgets**

677 Apertis widgets can be subclassed by system integrators in variants and used  
678 by application developers by creating shared libraries linking to the Apertis  
679 widget library. Applications then link to said new library and once the new  
680 widgets are registered with the `GObject` type system they can be referred to

681 in ClutterScript user interface files. If a system integrator wants a radically  
682 different widget, they can write something from scratch, ensuring to implement  
683 the appropriate interface. Subclassing existing widgets is for convenience but  
684 not technically necessary.

685 Apertis widgets should be as modularised as possible, splitting functionality into  
686 virtual methods where a system integrator might want to override it. For exam-  
687 ple, if a system integrator wants the roller widget to have a different activation  
688 animation depending on the number of items in the model, they could create  
689 a roller widget subclass, and override the appropriate virtual methods (in this  
690 case activate) and update the animation as appropriate:

---

```

1  G_DEFINE_TYPE (MyRoller, my_roller, LIGHTWOOD_TYPE_ROLLER)
2  static void
3  my_roller_init (MyRoller *self)
4  {
5  }
6
7  static gboolean
8  my_roller_activate (MyRoller *self,
9  gint item_id)
10 {
11     LightwoodRoller *roller;
12     LightwoodRollerClass *roller_class;
13     LightwoodModel *model;
14
15     roller = LIGHTWOOD_ROLLER (self);
16     roller_class = LIGHTWOOD_ROLLER_GET_CLASS (roller);
17     model = lightwood_roller_get_model (roller);
18
19     if (lightwood_model_get_n_items (model) > 5) {
20         /* change animation */
21     } else {
22         /* reset animation */
23     }
24
25     /* chain up */
26     return roller_class->activate (roller, item_id);
27 }
28
29 static void
30 my_roller_class_init (MyRollerClass *klass)
31 {
32     LightwoodRollerClass *roller_class = LIGHTWOOD_ROLLER_CLASS (klass);
33
34     roller_class->activate = my_roller_activate;
35 }

```

---

691 Another example is if the system integrator wants to change another part of the  
692 roller when scrolling starts, the appropriate signal can be connected to:

```

693 G_DEFINE_TYPE (MyRoller, my_roller, LIGHTWOOD_TYPE_ROLLER)
694
695 static void
696 my_roller_scrolling_started (MyRoller *self,
697                             gpointer user_data)

```

```

698 {
699     /* scrolling has started here */
700 }
701
702 static void
703 my_roller_constructed (GObject *obj)
704 {
705     /* chain up */
706     G_OBJECT_GET_CLASS (obj)->constructed (obj);
707     g_signal_connect (obj, "scrolling-started", G_CALLBACK (my_roller_scrolling_started), NULL);
708 }
709
710 static void
711 my_roller_class_init (MyRollerClass *klass)
712 {
713     GObjectClass *object_class = G_OBJECT_CLASS (klass);
714     object_class->constructed = my_roller_constructed;
715 }

```

716 In the template, this variant would stop referring to `LightwoodRoller` and instead  
717 would use `MyRoller`, or update the widget factory entry (see [Widget factories](#)).

## 718 Models

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

722 The actual implementation of the model should be of no importance to the  
723 widgets, and only basic model interface methods should be called by any widget.  
724 It is suggested to use the [GListModel](#)<sup>7</sup> interface as said model interface as it  
725 provides a set of simple type-safe methods to enumerate, manipulate, and be  
726 notified of changes to the model.

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

## 729 Theming

730 Using the `GtkStyleContext` object from GTK+ is wise for styling widgets as it can  
731 aggregate styling information from many sources, including CSS. GTK+'s CSS  
732 parsing code is advanced and well tested as GTK+ itself switched its [Adwaita](#)<sup>8</sup>  
733 default theme to pure CSS some time ago, replacing theme engines that required  
734 C code to be written to customise appearance.

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

<sup>8</sup><https://git.gnome.org/browse/gtk+/tree/gtk/theme/Adwaita>

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

## 740 Clutter Widgets

741 The `GtkApertisStylable` interface is a mixin for any `GObject` to enable use of a  
742 `GtkStyleContext`. It is an Apertis-specific interface and therefore not candidate  
743 for upstreaming, and will need maintaining as the CSS machinery in GTK+  
744 changes over time.

745 Every existing Clutter widget will have to be manually taught to use the style  
746 context and any special requirements will also need to be applied from the style  
747 context as necessary.

## 748 Theme Changes

749 Applications should listen to a documented [GSettings](#)<sup>9</sup> key for changes to the  
750 theme and icon theme. Changes to the theme should update the style properties  
751 in the `GtkStyleContext` and will trigger a widget redraw and changes to the icon  
752 theme should update the icon paths and trigger icon redraws.

## 753 Language Support

754 GNU [gettext](#)<sup>10</sup> is a well-known system for managing translations of applications.  
755 It provides tools to scan source code looking for translatable strings and a library  
756 to resolve said strings against language files which are easily updated without  
757 touching the source code of said applications.

## 758 Language Changes

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

## 761 Updating Languages

762 Language files for GNU `gettext` saved into the appropriate directory can be  
763 easily used immediately with no other changes to the application. Over the  
764 air (OTA) updates can contain updated language files which get saved to the  
765 correct location and would be loaded the next time the application is started.

---

<sup>9</sup><https://developer.gnome.org/gio/stable/GSettings.html>

<sup>10</sup><https://www.gnu.org/software/gettext/>



## 766 Day & Night Mode

767 Inspired by GTK+'s *dark mode*<sup>11</sup>, variant CSS should provide a `dark` class for  
768 widgets to be used in night mode. If the `dark` class is not set the user interface  
769 should be in day mode. *CSS transitions*<sup>12</sup> should make the animation smooth.

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

## 772 View Management

773 At the time of writing, Clutter does not have a built-in view management sys-  
774 tem. GTK+ has `GtkStack`<sup>13</sup> for managing views and displaying animations  
775 when moving between one view and another. The useful parts of `GtkStack` could  
776 be migrated to Clutter (subject to suitable licensing) to re-use the functionality  
777 and user testing and not waste effort in reimplementing everything from scratch.  
778 Existing view management systems (for example, in `libthornbury`) should also  
779 be considered for this migration task.

## 780 Speed Lock

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

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

## 787 Apertis Widgets

788 Applications that use Apertis widgets extensively should have very little to  
789 modify to support the speed lock. Apertis widgets should read and monitor the  
790 GSettings key to change their content when necessary:

- 791 • Lists with kinetic scrolling – disable the kinetic scrolling (see [Scrolling](#)  
792 [lists](#)).
- 793 • Text – very long texts in text views or label widgets should be hidden if  
794 there is no alternative provided (see [Text](#)).
- 795 • Keyboard – do not show the keyboard and provide feedback to the appli-  
796 cation as to whether the keyboard could appear or not (see [Keyboard](#)).
- 797 • Pictures – mast the picture shown in the picture widget (see [Pictures](#)).

---

<sup>11</sup><https://developer.gnome.org/gtk3/stable/GtkSettings.html#GtkSettings--gtk-application-prefer-dark-theme>

<sup>12</sup>[http://www.w3schools.com/css/css3\\_animations.asp](http://www.w3schools.com/css/css3_animations.asp)

<sup>13</sup><https://developer.gnome.org/gtk3/stable/GtkStack.html>

- 798 • Video playback – either pause the video completely or just mask the video  
799 and keep the audio sounding (see [Video playback](#)).
- 800 • Map – disable the kinetic scrolling (see [Map gestures](#)).
- 801 • Web view – mask the contents entirely (see [Web view](#)).

802 Apertis widgets should fill text widgets with contents that can differ depending  
803 on whether the speed lock is active or not.

#### 804 **List Columns**

805 The number of columns visible should be reduced to remove superfluous infor-  
806 mation when the speed lock is active (see [List columns](#)). The nature of every  
807 list can be different and the detection of superfluous information is impossible  
808 automatically. There should be a way of either application authors specifying  
809 which columns should be hidden, or it should be left up to the application itself.  
810 If the latter is not an option (see enforcement comments in [Speed lock](#)), the  
811 entire list widget should be masked to hide its contents.

#### 812 **Keyboard**

813 As mentioned in [Keyboard](#), applications should deal with the possibility that  
814 the keyboard may not be available at any given time, if the speed lock is active.  
815 In the case that the keyboard request is denied, the application should change  
816 its user experience slightly to accommodate for this, such as the example with  
817 bookmarks given previously.

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

#### 821 **Templates**

822 Apertis-provided templates should have versions for when the speed lock is  
823 activated and Aperis widgets should switch to these templates accordingly.

#### 824 **Insensitive Widgets**

825 As highlighted in [Insensitive widgets](#), it should be made obvious to the user when  
826 functionality is disabled, and why. There should be a uniform visual change  
827 to widgets when they have been made insensitive so users can immediately  
828 recognise what is happening.

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

## 831 **Notifications**

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

## 835 **Masking Unknown Applications**

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

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

## 843 **References**

### 844 **GTK+ Migration**

845 In an older version of this document it was posed that a move from Clutter to  
846 GTK+ might be wise. It has been decided that for the time being a move is  
847 unwise due to the immature nature of the GTK+ Scene Graph Kit.

848 The following sections were removed from the previous sections in this document  
849 and have been left here for future reference.

### 850 **GTK+ or Clutter**

851 The following suggestions are possible using either the GTK+ or Clutter li-  
852 braries. Existing code is currently written in Clutter, but a move to GTK+  
853 could be wise because GTK+ is still highly used and maintained, whereas Clut-  
854 ter is less used and less maintained. The maintainers of Clutter have even  
855 announced that planned future additions to GTK+ would [deprecate Clutter](#)<sup>14</sup>.  
856 Although not in stone, the deprecation is [planned](#)<sup>15</sup> for the 3.20 release of GTK+  
857 which is planned in March 2016.

858 It is worth noting that Clutter widgets can be embedded inside GTK+ appli-  
859 cations, and GTK+ widgets can be embedded inside Clutter applications, but  
860 there are many problems with input and ensuring GTK+ functions are only  
861 called from GTK+ callbacks, so following this path is likely not worth the even-  
862 tual problems.

863 For completeness, the following sections with toolkit-specific approaches are  
864 split into two such that both GTK+ and Clutter paths can be considered.

---

<sup>14</sup><https://www.bassi.io/articles/2014/07/29/guadec-2014-gsk/>

<sup>15</sup><https://wiki.gnome.org/Projects/GTK%2B/Roadmap>

## 865 Specification in GtkBuilder

866 GtkBuilder is a method for creating user interfaces from XML files. An example  
867 is shown below which describes the A-variant application chooser user interface:

```
868 <interface>
869   <object class="LightwoodAppCategoryModel" id="model_categories" />
870   <object class="LightwoodAppModel" id="model_apps" />
871   <object class="LightwoodWindow" id="window">
872     <child>
873       <object class="GtkBox" id="hbox1">
874         <property name="homogeneous">True</property>
875         <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
876         <child>
877           <object class="LightwoodRoller" id="roller_categories">
878             <property name="model">model_categories</property>
879             <property name="app-list">roller_apps</property>
880             <signal name="activated" handler="category_activated_cb" />
881           </object>
882         </child>
883         <child>
884           <object class="LightwoodRoller" id="roller_apps">
885             <property name="model">model_apps</property>
886             <signal name="activated" handler="app_activated_cb" />
887           </object>
888         </child>
889       </object>
890     </child>
891   </object>
892 </interface>
```

893 The first two objects created (`model_categories` and `model_apps`) are models for  
894 the application categories available on the system, and the applications avail-  
895 able on the system—due to their class names (`LightwoodAppCategoryModel` and  
896 `LightwoodAppModel` respectively). These models are not widgets visible in the  
897 user interface, but proper widgets will refer to them later in the template.

898 The next entry describes the main window in the user interface, inside of which  
899 there is a horizontal box (with some style properties set), with two children that  
900 are both of type `LightwoodRoller`. Although these widgets are of the same type,  
901 they are different instances and they have been given different models. The first  
902 roller widget (on the left) has been given the `model_categories` model and the  
903 second roller widget (on the right) has been given the `model_apps` model, both  
904 created at the beginning of the XML file.

905 Additionally the `LightwoodRoller::activated` signal is connected on both rollers  
906 to different callbacks. The signal callback names are listed in the application  
907 documentation. In this case, when the left-hand roller with categories is changed

908 (activated), the right-hand roller with applications is updated (set by the `app-`  
909 `list` property).

910 Another example to compare is given below with the B-variant application  
911 chooser user interface:

```
912 <interface>
913   <object class="LightwoodAppModel" id="model_apps" />
914   <object class="LightwoodWindow" id="window">
915     <child>
916       <object class="LightwoodList" id="list_apps">
917         <property name="model">model_apps</property>
918         <signal name="activated" handler="app_activated_cb" />
919       </object>
920     </child>
921   </object>
922 </interface>
```

923 The differences of the B-variant application chooser in comparison to the A-  
924 variant application chooser are:

- 925 1. There is no categories model and no categories roller.
- 926 2. There is no more box inside the main window widget.
- 927 3. The list widget is a `LightwoodList` instead of a `LightwoodRoller`. This is  
928 a visual difference dictated by the widget implementation and chosen for  
929 this variant, but the data backend for both lists (in the `model_model_apps`)  
930 is unchanged.

931 These are just two examples of how an application chooser could be designed.  
932 The user interface files contain minimal theming as that is achieved in separate  
933 CSS files (see [Theming](#)).

934 Typically, applications will come with many templates for system integrators to  
935 either use, or take guidance from.

## 936 **GTK+ Widgets**

937 Support for `GtkStyleContext` inside GTK+ widgets is already present. Widgets  
938 inside the GTK+ library (and therefore also their subclasses) already talk to  
939 the style context and are drawn according to custom styling.

940 New GTK+ widgets with special requirements would need to get the appropriate  
941 style information from the style context and apply it as necessary. This is  
942 documented in the GTK+ documentation and it is easy to find examples of it  
943 in the source code.

944 **Appendix**

945 **Variant Differences**

946 **Thumbnail View**



947



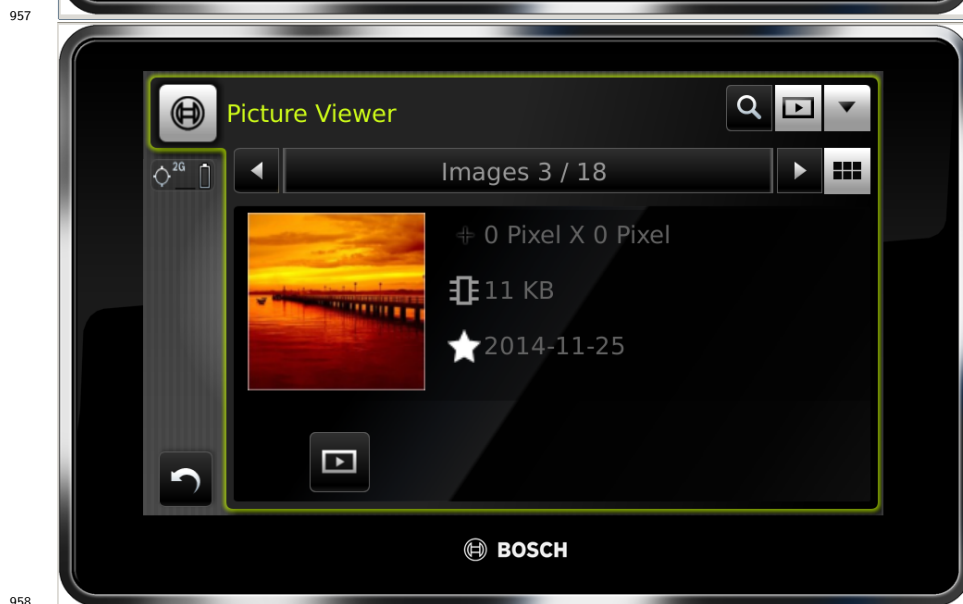
948

949 • Re-used:

950 – Roller

- 951           – Views drawer
- 952       • Differences:
- 953           – In variant A, one needs to go back to the app launcher to start the
- 954           photo viewer with tags/title/date as they are all separate apps; in B
- 955           it is in the same app.

956 **Detail View**



958

- 959 • Re-used: nothing
- 960 • Differences:
  - 961 – In variant A it is a roller; in B it is an individual image.
  - 962 – In variant B there is a media info widget; in A it's a roller on the
  - 963 right.

964 **List View**



965





966

967

968

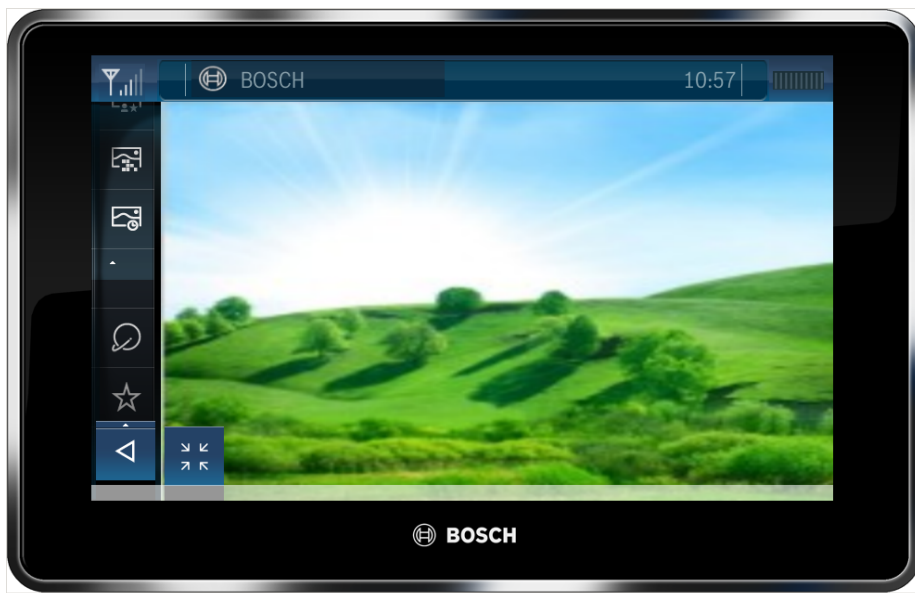
969

970

- Re-used: nothing
- Differences:
  - The roller is different (it displays different information).
  - In variant B one can delete; in A the feature is not present.

971

**Full Screen**



972



973

974

- Re-used: nothing

975

- Differences:

976

– The full screen is a roller in variant A; in B it is a single snapshot.

977

– In variant B there are many extra functions; in A these functions are

978

not present.