



Internationalization

Contents

2	Internationalization	2
3	Text input	2
4	Text display	4
5	UI layout	7
6	Localization	8
7	Translation	8
8	Testing	13
9	Other locale configuration	14
10	Distribution	14
11	Runtime switching of locale	15
12	Common pattern	15
13	Application helper API	17
14	Localization in GNOME	17

This design explains how the Apertis platform will be made localizable and how it will be localized to specific locales.

“Internationalization”(“i18n”) is the term used for the process of ensuring that a software component can be localized. “Localization”(“l10n”) is the process of adding the necessary data and configuration so an internationalized software adapts to a specific locale. A locale is the definition of the subset of a user’s environment that depends on language and cultural conventions.

All this will be done with the same tools used by GNOME and we do not anticipate any new development in the middleware itself, though UI components in the Apertis shell and applications will have to be developed with internationalization in mind, as explained in this document.

For more detailed information of how translation is done in the FOSS world, a good book on the subject is [available](#)¹.

Internationalization

Text input

Some writing systems will require special software support for entering text, the component that provides this support for an specific writing system is called input method. There is a framework for input methods called [IBus](#)² that is the most common way of providing input methods for the different writing systems. Several input methods based on IBus are available in Ubuntu, and it is very unlikely that any needs will not be covered by them. An older, but more broadly-supported, input method framework is [SCIM](#)³ and an even older one is

¹<http://archive.flossmanuals.net/open-translation-tools/>

²http://en.wikipedia.org/wiki/Intelligent_Input_Bus

³http://en.wikipedia.org/wiki/Smart_Common_Input_Method

37 [XIM](#)⁴.

38 The advantage of using an input method framework (instead of adding the func-
39 tionality directly to applications or widget libraries) is that the input method
40 will be usable in all the toolkits that have support for that input method frame-
41 work.

42 Note that currently there is almost no support in Clutter for using input meth-
43 ods. Lead Clutter developer Emmanuele Bassi recommends doing something
44 similar to GNOME Shell, which uses [GtkIMContext](#)⁵ on top of [ClutterText](#)⁶, which
45 would imply depending on GTK+. There's a project called clutter-imcontext
46 that provides a simple version of GtkIMContext for use in Clutter applications,
47 but Emmanuele strongly discourages its use. GTK+ and Qt support XIM,
48 SCIM and IBus.

49 In order to add support for GtkIMContext to ClutterText, please see how it's
50 done in [GNOME Shell](#)⁷. As can be seen this implementation calls the following
51 functions from the [GtkIMContext](#)⁸ API:

```
52     • gtk_im_context_set_cursor_location  
53     • gtk_im_context_reset  
54     • gtk_im_context_set_client_window  
55     • gtk_im_context_filter_keypress  
56     • gtk_im_context_focus_in  
57     • gtk_im_context_focus_out
```

58 Between the code linked above and the GTK+ API reference it should be rea-
59 sonably clear how to add GtkIMContext support to Clutter applications, but
60 there's also the possibility of reusing that code instead of having to rewrite it.
61 In that case, we advise to take into account the license of the file in question
62 (LGPL v2.1).

63 For systems without a physical keyboard, text can be entered via a virtual key-
64 board. The UI toolkit will invoke the on-screen keyboard when editing starts,
65 and will receive the entered text once it has finished. So the on-screen key-
66 board can be used for text input by a wide variety of UI toolkits, Collabora
67 recommends it to use IBus.

68 The reasons for recommending to use an input-method framework is that most
69 toolkits have support for it, so if an application is reused that uses Qt, the on-
70 screen keyboard will be used without any specific modification, which wouldn't
71 be the case if `GtkIMContext` would be used.

⁴<http://www.x.org/releases/X11R7.6/doc/libX11/specs/XIM/xim.html>

⁵<http://developer.gnome.org/gtk/unstable/GtkIMContext.html#GtkIMContext.description>

⁶<https://developer.gnome.org/st/stable/StEntry.html>

⁷<http://git.gnome.org/browse/gnome-shell/tree/src/st/st-im-text.c>

⁸<http://developer.gnome.org/gtk/unstable/GtkIMContext.html#GtkIMContext.description>

72 About why to use IBus over other input-method frameworks, the reason is that
73 IBus is already supported by most modern toolkits, has a very active upstream
74 community and the cost of developing input-methods with IBus is lower than
75 with other frameworks. Currently, IBus is the default input method framework
76 in Ubuntu and Fedora, and GNOME is considering dropping support for other
77 frameworks'input methods.

78 **Text display**

79 For text layout and rendering the toolkit needs to support all writing systems we
80 are interested in. GTK+ and Clutter use Pango which supports a very broad
81 set of natural language scripts. The appropriate fonts need to be present so
82 Pango can render text.

83 The recommended mechanism for translating those pieces of text that are dis-
84 played in the UI is to export those strings to a file, get them translated in
85 additional files and then have the application use at runtime the appropriate
86 translated strings depending on the current locale. GNU gettext implements
87 this scheme and is very common in the FOSS world. Gettext also allows adding
88 a comment to the string to be translated, so it gives more context that can aid
89 the translator to understand better how the string is used in the UI. This ad-
90 ditional context can also be used to encode additional information as explained
91 later. The GNU [gettext](http://www.gnu.org/software/gettext/manual/gettext.html)⁹ manual is comprehensive and covers all this in detail.

92 This is an example of all the metadata that a translated string can have attached:

```
93 #. Make sure you use the IEC equivalent for your language
94 ## Have never seen KiB used in our language, so we'll use KB
95 #: ../glib/gfileutils.c:2007
96 #, fuzzy, c-format
97
98 msgctxt "File properties dialog"
99 msgid "%.1f KiB"
100 msgstr "%.1f KB"
```

101 For strings embedded inside [ClutterScript] files, ClutterScript supports a `trans-`
102 `latable` property to mark the string as translatable. So to mark the text of a
103 `ClutterText` as translatable, the following ClutterScript should be used:

```
1  "label" : {
2      "text" : {
3          "translatable" : true,
4          "string" : "Label Text"
5      }
6  }
```

⁹<http://www.gnu.org/software/gettext/manual/gettext.html>

104 Note that `clutter_script_set_translation_domain()` or `textdomain()`¹⁰ needs to
105 be called before translatable strings can be used in a ClutterScript file.

106 `gettext`¹¹ currently does not support extracting strings from ClutterScript files;
107 support for that needs to be added.

108 Previous versions of this document recommended using `intltool`¹². However,
109 in recent years, it has been superseded by `gettext`¹³. Previously, `gettext` was
110 unmaintained, and `intltool` was developed to augment it; now that `gettext` is
111 actively maintained and gaining new features, `intltool` is no longer necessary.

112 **Message IDs** It is most common in FOSS projects (specially those using GNU
113 `gettext`) to use the English translation as the identifier for the occurrence of a
114 piece of text that needs to be translated, though some projects use an identifier
115 that can be numeric (`T54237`) or a mnemonic (`PARK_ASSIST_1`). The IDs will not
116 leak to the UI if the translations are complete, and there is also the possibility
117 of defining a fallback language.

118 There's two main arguments used in favor of using something other than plain
119 English as the ID:

- 120 • so that when the English translation is changed in a trivial way, that
121 message isn't marked as needing review for all other languages;
- 122 • and to avoid ambiguities, as "Stop" may refer to an action or a state and
123 thus may be translated differently in some languages, while using the IDs
124 `state_stop` and `action_stop` would remove that ambiguity.

125 When using `gettext`, the first argument loses some strength as it includes a tool
126 that is able to merge the new translatable string with the existing translations,
127 but marking them as in need of review. About the argument of avoiding am-
128 biguity, GNU `gettext` was extended to provide a way of attaching additional
129 context to a message so that is not a problem anymore.

130 Regarding advantages of using plain English (or other natural language) as the
131 message ID:

- 132 • better readability of the code,
- 133 • when the developers add new messages to the application and run it, they
134 will see the English strings which is closer to what the user will see than
135 any other kind of IDs.

136 From the above it can be understood why it's normally recommended to just
137 use the English translation as the placeholder in the source code when using
138 GNU `gettext`.

¹⁰<https://manpages.debian.org/unstable/gettext-base/textdomain.3.en.html>

¹¹<http://www.gnu.org/software/gettext/manual/gettext.html>

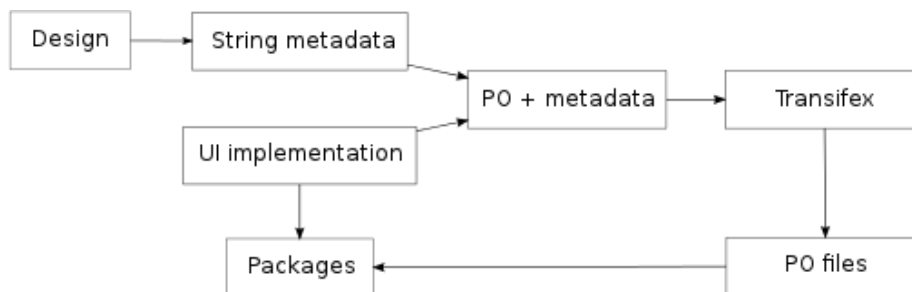
¹²<https://launchpad.net/intltool/>

¹³<http://www.gnu.org/software/gettext/manual/gettext.html>

139 Regarding consistency, there's a slight advantage in using natural language
 140 strings because when entering translations the translation software may offer
 141 suggestions from the translation memory and given that the mnemonic IDs are
 142 likely to be unique, there will be less exact matches.

143 Because of the need to associate to each translation metadata such as the font
 144 size and the available space, plus having product variants that share most of
 145 the code but can have differences in fonts and widget sizes, we recommend to
 146 use mnemonics as IDs, which would allow us to keep a list of the translatable
 147 strings and their associated fonts and pixels for each variant. This will be further
 148 discussed in [Testing](#).

149 This diagram illustrates the workflow that would be followed during localization.



150
 151 For better readability of the source code we recommend that the IDs chosen
 152 suggest the meaning of the string, such as *PARK_ASSIST_1*. Instead of hav-
 153 ing to specify whole font descriptions for each string to translate, Collabora
 154 recommends to use styles that expand to specific font descriptions.

155 Here is an example of such a metadata file, note the font styles `NORMAL`, `TITLE`
 156 and `APPLICATION_LIST`:

```

157 PARK_ASSIST_1 NORMAL 120px
158 PARK_ASSIST_2 NORMAL 210px
159 SETTINGS_1 TITLE 445px
160 BROWSER APPLICATION_LIST 120px
  
```

161 And here is the PO file that would result after merging the metadata in, ready
 162 to be uploaded to Transifex:

```

163 #. NORMAL,120px
164 #: ../preferences.c:102
165 msgid "PARK_ASSIST_1"
166 msgstr "Park assist"
167 #. NORMAL,210px
168 #: ../preferences.c:104
169 msgid "PARK_ASSIST_2"
170 msgstr "Park assist"
  
```

171 If for some reason some source code is reused that uses English for its translation

172 IDs and the rest of the application or library uses synthetic IDs, Collabora
173 recommends to have a separate domain for each section of the code, so all
174 English IDs are in their own PO file and the synthetic IDs in their own. In this
175 case, note that matching metadata to individual strings can be problematic if
176 the metadata isn't updated when the string IDs change. It will be a problem as
177 well if there are several occurrences of exactly the same string.

178 When it is needed to modify the metadata related to existing strings, the process
179 consists of modifying the file containing string metadata, then merging it again
180 with the PO files from the source code and importing it into the translation
181 management system.

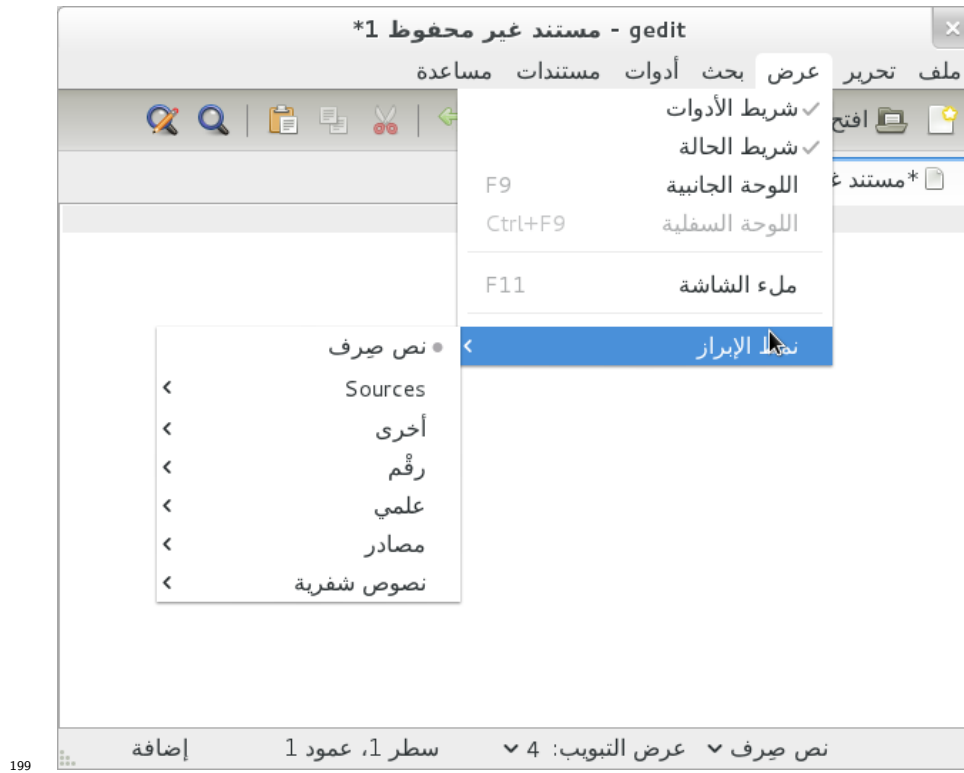
182 **Consistency** Translation management systems offer tools to increase the con-
183 sistency of the translations, so the same words are used to explain the same
184 concept. One of the tools that Transifex offers is a search feature that allows
185 to quickly check how a word has been translated in other instances. Another is
186 the *translation memory* feature, which suggests translations based on what has
187 been translated already.

188 There isn't any relevant difference in how these tools work and whether the
189 strings are identified by synthetic IDs or by their English translations.

190 **UI layout**

191 Some languages are written in orientations other than left to right and users
192 will expect that the UI layout takes this into account. This means that some
193 horizontal containers will have to layout its children in reverse order, labels
194 linked to a widget will also be mirrored, and some images used in icons will
195 have to be mirrored horizontally as well.

196 Here is an example of an application running under a locale whose orientation
197 is right-to-left, note the alignment of icons in the toolbar and the position of
198 the arrows in submenus:



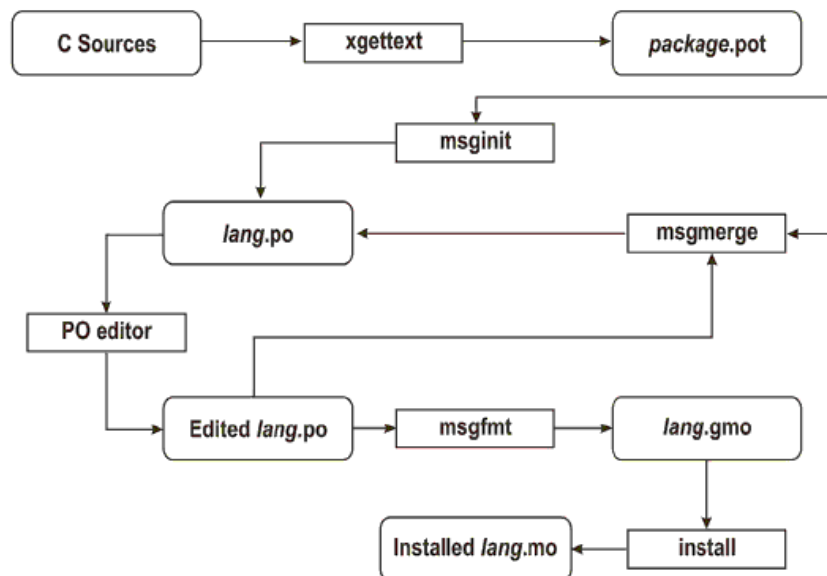
200 Localization

201 Translation

202 **GNU gettext** Most of the work happens in the translation phase, in which
 203 .po files are edited so they contain appropriate translations for each string in the
 204 project. As illustrated in the diagram below, the .po files generated from the
 205 original .pot file serve as the basis for starting the translation. When the source
 206 code changes and thus a different .pot file gets generated, GNU gettext includes
 207 a tool for merging the new .pot file into the existing .po files so translators can
 208 work on the latest code.

209 This diagram illustrates the [workflow](http://upload.wikimedia.org/wikipedia/commons/0/05/GNU_gettext_process.png)¹⁴ when using GNU gettext to translate
 210 text in an application written in C:

¹⁴http://upload.wikimedia.org/wikipedia/commons/0/05/GNU_gettext_process.png



GNU gettext Working Process

211

212 From time to time, it is needed to extract new translatable strings from the
 213 source code and update the files that are used by translators. The extraction
 214 itself is performed by the tool [xgettext](#)¹⁵, which generates a new POT file con-
 215 taining all the translatable strings plus their locations in the source code and
 216 any additional context.

217 These are the [programming languages](#)¹⁶ supported by GNU gettext: C, C++,
 218 ObjectiveC, PO, Python, Lisp, EmacsLisp, librep, Scheme, Smalltalk, Java,
 219 JavaProperties, C#, awk, YCP, Tcl, Perl, PHP, GCC-source, NXStringTable,
 220 RST and Glade.

221 The POT file and each PO file are fed to [msgmerge](#)¹⁷ which merges the exist-
 222 ing translations for that language into the POT file. Strings that haven't been
 223 changed in the source code get automatically merged and the remaining are
 224 passed through a fuzzy algorithm that tries to find the corresponding translat-
 225 able string. Those strings that had a fuzzy match are marked as needing review.
 226 If strings are indexed with unique IDs instead of the English translation, then
 227 it's recommended to use the `-no-fuzzy-matching` option to msgmerge, so new

¹⁵http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html_node/xgettext-Invocation.html

¹⁶http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html_node/xgettext-Invocation.html#index-supported-languages_002c_0040cod

¹⁷http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html_node/msgmerge-Invocation.html

228 IDs will be always empty. Otherwise, if the POT file contained already an en-
229 try for `PARK_ASSIST_1` and `PARK_ASSIST_2` was added, when merging into existing
230 translations, the existing translation would be reused, but marking the entry as
231 fuzzy (which would cause Transifex to use that translation as a suggestion).

232 **Translation management** Though these file generation steps can be exe-
233 cuted manually with command line tools and translators can work directly on
234 the `.po` files with any text editor, there are more high-level tools that aim to
235 manage the whole translation process. Next we briefly mention the ones most
236 commonly used in FOSS projects.

237 Pootle¹⁸, Transifex¹⁹ and Launchpad Rosetta²⁰ are tools which provide conve-
238 nient UIs for translating strings. They also streamline the process of translating
239 strings from new `.pot` versions and offer ways to transfer the resulting `.po` files
240 to source code repositories.

241 Pootle is the oldest web-based translation management system and is mature
242 but a bit lacking in features. Maintaining an instance requires a fair amount of
243 experience.

244 Transifex is newer and was created to accommodate better than Pootle to the
245 actual workflows of most projects today. Its UI is richer in features that facilitate
246 translation and, more importantly, has good commercial support (by Indifex).
247 It provides as well an API that can be used to integrate it with other systems.
248 See [Transifex](#) for more details.

249 Launchpad is not easily deployable outside launchpad.net and is very oriented
250 to Ubuntu's workflow, so we do not recommend its usage.

251 Both Pootle and Transifex have support for translation memory, which aids in
252 keeping the translation consistent by suggesting new translations based on older
253 ones.

254 If for some reason translators prefer to use a spreadsheet instead of web UIs or
255 manually editing the PO files, [csv2po](#)²¹ will convert a PO file to a spreadsheet
256 and will convert it back so the translation system can be refreshed with the new
257 translations.

258 *po2csv* will convert a PO file to a CSV one which has a column for the comments
259 and context, another for the *msgid* and one more for the translation for the given
260 language. *csv2po* will do the opposite conversion.

261 It's very likely that the CSV format that these tools generate and expect doesn'
262 t match exactly what it is needed, so an additional step will be needed that

¹⁸<http://pootle.translatehouse.org/?id=pootle/index>

¹⁹<https://www.transifex.com/>

²⁰<https://translations.launchpad.net/>

²¹<http://docs.translatehouse.org/projects/translate-toolkit/en/latest/commands/csv2po.html?id=toolkit/csv2po>

263 converts the CSV file to the spreadsheet format required, and a step that does
264 the opposite.

265 **Transifex** In this section we discuss in more details some aspects of Transifex.
266 For an overview on other features of Transifex, please see the documentation
267 for [management](#)²² and [translation](#)²³.

268 **Deployment options** Transifex is available as a hosted web service in
269 <http://www.transifex.com>²⁴ and there are

270 several [pricing options](#)²⁵ depending on the project size, features and level of
271 technical support desired.

272 The FOSS part of Transifex is available as Transifex Community Edition
273 and can be freely downloaded and installed in any machine with a mini-
274 mally modern and complete Python installation. This version lacks some
275 of the features that are available in <http://transifex.com>²⁶ and in the En-
276 terprise Edition. The installation manual for the community edition is in
277 <http://help.transifex.net/server/install.html>.

278 The hosted and the enterprise editions support these features in addition of
279 what the community edition supports:

- 280 • Translation memory
- 281 • Glossary
- 282 • Improved collaboration between translators
- 283 • Improved UI theme

284 The advantage of the hosted edition is that it is updated more frequently
285 (weekly) and that in the future it will be possible to order paid translations
286 through the platform.

287 Transifex currently cannot estimate the space that a given translation will take
288 and will need to be extended in this regard.

289 It also fully supports using synthetic translation IDs instead of English or other
290 natural language.

291 Finally, Indifex provides commercial support for the enterprise edition of Tran-
292 sifex, which can either be self-hosted or provided as SaaS. Their portfolio in-
293 cludes assistance with deployment, consultancy services on workflow and cus-
294 tomization, and a broad package of technical support.

²²<https://www.transifex.com/features/team-management/>

²³<https://www.transifex.com/features/translation-tools/>

²⁴<http://www.transifex.com/>

²⁵<https://www.transifex.com/pricing/>

²⁶<http://transifex.com/>

295 **Maintenance** Most maintenance is performed through the web interface, by
296 registered users of the web service with the appropriate level of access. This
297 includes setting up users, teams, languages and projects. Less frequent tasks
298 such as instance configuration, software updates, performance tuning and set
299 up of automatic jobs are performed by the administrator of the server hosting
300 the service.

301 **Translation memory** Transifex will provide suggestions when translating a
302 string based on existing [translations](#)²⁷ in the current module or in other modules
303 that were configured to share their translation [memory](#)²⁸. This memory can also
304 be used to pre-populate translations for a new module based on other modules'
305 [translations](#)²⁹.

306 **Glossary** Each project has a series of terms that are very important to trans-
307 late consistently or that can have several different possible translations with
308 slightly different meanings. To help with this, Transifex provides a [glossary](#)³⁰
309 that will assist translators in these cases.

310 **POT merging** As explained in [GNU Gettext](#), new translatable strings are
311 extracted from the source files with the tool `xgettext` and the resulting POT file
312 is merged into each PO file with the tool `msgmerge`.

313 Once the PO files have been updated, the tool `tx` (command-line transifex client)
314 can be used to submit the changes to the server, this merge happening as [fol-](#)
315 [lows](#)³¹:

316 Here's how differences between the old and new source files will be handled:

- 317
- New strings will be added.
 - Modified strings will be considered new ones and added as well.
 - Strings which do not exist in the new source file (including ones which
320 have been modified) will be removed from the database, along with their
321 translations.

322 Keep in mind, however, that old translations are kept in the Translation Memory
323 of your project.

324 Note that this process can be automated.

325 **Automatic length check** Transifex's database model will have to be updated
326 to store additional metadata about each string such as the font description and
327 the available size in pixels. The web application could then check how many

²⁷<https://www.transifex.com/features/translation-tools/>

²⁸<https://docs.transifex.com/translation-memory/sharing-tm>

²⁹<https://docs.transifex.com/translation-memory/enabling-autofill/>

³⁰<https://docs.transifex.com/glossary/glossary>

³¹<https://docs.transifex.com/client/push>

328 pixels the entered string would take in the UI, using Pango and [Fontconfig](http://fontconfig.org/)³².
329 For better accuracy, the exact fonts that will be used in the UI should be used
330 for this computation.

331 Alternatively, there could be a extra step after each translation phase that would
332 spot all the strings that may overflow and mark them as needing review.

333 Testing

334 Translations will be generally proof-read, but even then we recommend testing
335 the translations by running the application to catch a number of errors which
336 are noticeable only at run time. This run-time evaluation can spot confusing or
337 ambiguous wording, as well as layout problems.

338 Each translation of a single piece of text can potentially require a wildly-differing
339 width due to varying word and expression sizes in different languages. There
340 are ways for the UI to adapt to the different string sizes but there are limits
341 to how well this can work, so translators need often to manually check whether
342 their translation fits nicely in the UI.

343 One way to automatically avoid many instances of layout errors would be to have
344 available, during translation and along with the extracted strings, the available
345 space in pixels and the exact font description used to display the string. This
346 information would allow automatic calculation of string sizes, thus being able to
347 catch translations that would overflow the boundaries. As explained in [Message](#)
348 [IDs](#), this metadata would be stored in a file indexed by translation ID and would
349 be merged before importing it into the translation management software, which
350 could use it to warn when a translated string may be too long. For this to
351 consistently work, the translation IDs need to be unique (and thus synthetic).

352 When calculating the length of a translation for a string that contains one or
353 more [printf placeholders](#)³³, the width that the string can require when displayed
354 in the UI grows very quickly. For example, for the placeholder `%d` which can
355 display a 32-bit integer value, the final string can take up to 10 additional
356 digits. The only way to be safe is to assume that each placeholder can be
357 expanded to its maximum size, though in the case of strings (placeholder `%s`)
358 that is practically unlimited.

359 If, despite automatically warning the translator when a translation will not fit
360 in the UI, some strings are too long, the UI widget that displays the string could
361 ellipsize it to indicate that the displayed text isn't complete. If this occurred
362 in a debug build, a run-time warning could be also emitted. These warnings
363 would be logged only once a translated string has been displayed in the UI and
364 wouldn't apply to text coming from an external input.

365 For manual testing, an image could be provided to translators so they could
366 easily merge their work and test the software in their locale.

³²<http://fontconfig.org/>

³³<http://pubs.opengroup.org/onlinepubs/9699919799/functions/printf.html>

Other locale configuration

There is some other configuration that is specific to a locale but that is not specific to the application. This includes number, date and time formats, currency and collation. Most locales are already present in GNU glibc so we would only have to add a locale if it would target an extremely small population group.

Distribution

There are three main ways of packaging translations:

- package all the MO files (compiled PO files) along the rest of the files for a single component (for example `gnome-shell` in Ubuntu).
- package the MO files for a single component (usually a big one such as LibreOffice or KDE) and a specific language in a separate package (for example, `firefox-locale-de`³⁴ in Ubuntu).
- package several MO files corresponding to several components for one language (for example `language-pack-cs-base` in Ubuntu).

Our recommendation at this stage is to have:

- each application along with all its existing translations in a single package. This way the user will install e.g. `navigation-helper_1.10_armhf.deb` and the user will be able to switch between all the supported languages without having to install any additional packages.
- the rest of the MO files (those belonging to the UI that is pre-installed, such as applications and the shell) would be packaged grouped by language, e.g. `apertis-core-de_2.15_armhf.deb`. That way we can choose which languages will be pre-installed and can allow the user to install additional languages on demand.

If we do not want to pre-install all the required fonts and input methods for all supported languages, we could have meta-packages that, once installed, provide everything that is required to support a specific language. The meta-package in Ubuntu that provides support for Japanese is a good example of [this](#)³⁵.

Note that our current understanding is that the whole UI will be written, not reusing any existing UI components that may be present in the images. This implies that though some middleware components may install translations, those are not expected to be seen by the user ever.

This table should help make an idea of the sizes taken by packages related to localization:

Package name	Contents	Package size	Installed size
<code>language-pack-de-base</code>	MO files for core packages1	2,497 kB	8,432 kB

³⁴<http://packages.ubuntu.com/oneiric/firefox-locale-de>

³⁵<http://packages.ubuntu.com/hardy/language-support-ja>

Package name	Contents	Package size	Installed size
firefox-locale-de	German translation for Firefox ²	233 kB	453 kB
libreoffice-l10n-de	Resource files with translations, and templates ³	1,498 kB	3,959 kB
language-support-fonts-ja	Fonts for rendering Japanese	29,006 kB	41,728 kB
Ibus-anthy	Japanese input method ⁴	388 kB	1,496 kB

401 The *language-support-fonts-ja* package is a virtual one that brings the following
402 other packages (making up the total of 41,728 kB when installed):

Package name	Contents	Package size	Installed size
ttf-takao-gothic	Japanese TrueType font set, Takao Gothic Fonts	8,194.6 kB	12,076.0 kB
ttf-takao-pgothic	Japanese TrueType font set, Takao P Gothic Font	4,195.4 kB	6,196.0 kB
ttf-takao-mincho	Japanese TrueType font set, Takao Mincho Fonts	16,617.9 kB	23,456.0 kB

403 Modern distributions will bring all those fonts for Japanese-enabled installations,
404 but depending on the commercial requirements, a system could make with just
405 a subset. Similarly, other locales will require a set of fonts for properly rendering
406 text in the same way as users in specific markets expect. In order to recommend
407 specific font files, knowledge on the requirements are needed.

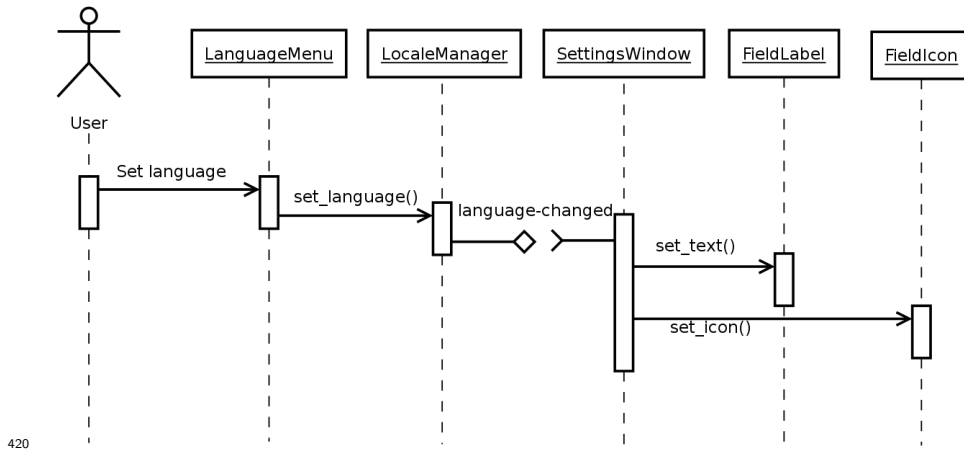
408 Runtime switching of locale

409 Common pattern

410 A usual way of implementing switching languages during runtime is to have
411 those UI components that depend on the language to listen for a signal that gets
412 emitted by a global singleton when the language changes. Those components
413 will check the new language and update strings and probably change layout if
414 the text direction has changed. Some other changes may be needed such as
415 changing the icons, colors, etc.

416 The Qt toolkit has a bit of support for this solution and their [documentation](#)³⁶
417 explains in detail how to implement it. This can be easily implemented in
418 Clutter and performance should be good provided that there isn't an excessive
419 amount of actors in the stage.

³⁶https://wiki.qt.io/How_to_create_a_multi_language_application



LocaleManager in the diagram would be a singleton that stores the current locale and notifies interested parties when it changes. The current locale would be changed by UI elements such as a combo-box in the settings panel, a menu option, etc.

Other UI elements that take locale-dependent decisions (in the diagram, SettingsWindow) would register to be notified when the locale changes, so they can change their UI (update strings, change icons, change text orientation, etc.).

Since systemd version 30, the [systemd-localed service](https://www.freedesktop.org/wiki/Software/systemd/localed/)³⁷ has been provided as a standard D-Bus API (`org.freedesktop.locale1`) for managing the system's locale, including being notified when it is changed, getting its current value, and setting a new value. This should be used in combination with the `org.gnome.system.locale` GSettings schema, which stores the *user's* locale preferences. We suggest that the LocaleManager from the diagram is implemented to query `org.gnome.system.locale` and returns the value of its `region` setting if set. If not set, the user is using the default system locale, which LocaleManager should query from `org.freedesktop.locale1`.

`org.freedesktop.locale1` is provided as a D-Bus API only, and `org.gnome.system.locale` is a GSettings schema. They are accessed differently, so a set of wrapper functions should be written as a convenience for application developers.

systemd-localed uses [polkit](https://www.freedesktop.org/wiki/Software/polkit/)³⁸ to authorise changes to the system locale, so vendors would need to write a policy which determines which applications are permitted to change the system locale, and which are allowed to query it. The default should be that only the system preferences application is allowed to change the locale; and all applications are allowed to query it (and be notified of changes to the locale).

³⁷<https://www.freedesktop.org/wiki/Software/systemd/localed/>

³⁸<https://www.freedesktop.org/wiki/Software/polkit/>

446 These snippets show how systemd-localed could be used by an application (omit-
447 ting asynchronous calls for simplicity):

448 The following example shows how the user's locale can be queried by
449 an application, first checking `org.gnome.system.locale`, then falling back to
450 `org.freedesktop.locale1` if the user is using the system locale. It is expected that
451 most of the code in this example would be implemented in the `LocaleManager`,
452 rather than being reimplemented in every application.

453 `{{ ../examples/locale-region-changed.c }}`

454 Application helper API

455 To reduce the amount of work that most application authors will have when
456 making their applications aware of runtime locale switches, we recommend that
457 the SDK API includes a subclass of `ClutterText` (let's call it `ExampleText`) that
458 reacts to locale changes.

459 `ExampleText` would accept a translatable ID via the function `example_text_set_text()`,
460 would display its translation based on the current locale and would also listen
461 for locale changes and update itself accordingly.

462 So `xgettext` can extract the string IDs that get passed to `ExampleText`, it would
463 have to be invoked with `--flag=example_text_set_text:1:c-format`.

464 If applications use `ExampleText` instead of `ClutterText` for the display of all their
465 translatable text, they will have to interface with `LocaleManager` only if they have
466 to localize other aspects such as icons or container orientation.

467 Localization in GNOME

468 GNOME uses a web application called Damned Lies to manage their translation
469 work-flow and produce statistics to monitor the translation progress. Damned
470 Lies is specifically intended to be used within GNOME, and its maintainers rec-
471 ommend other parties to look into a more generic alternative such as Transifex.
472 There used to be a separate tool called Vertimus but it has been merged into
473 Damned Lies.

474 Participants in the translation of GNOME belong to translation teams, one for
475 each language to which GNOME is translated, and they can have one of three
476 roles: translator, reviewer and committer. As explained in GNOME's [wiki](https://wiki.gnome.org/TranslationProject/ContributeTranslations)³⁹:

477 *Translators contains persons helping with GNOME translations into*
478 *a specific language, who added themselves to the translation team.*
479 *Translators could add comment to a specific PO file translation, could*
480 *reserve it for translations and could suggest new translations by up-*
481 *load a new PO file. The suggested translations will be reviewed by*
482 *other team members.*

³⁹<https://wiki.gnome.org/TranslationProject/ContributeTranslations>

483 *Reviewers are GNOME translators which were assigned by the team*
484 *coordinator to review newly suggested translations (by translators,*
485 *reviews or committers). They have access to all actions available to*
486 *a translators with the addition of some reviewing task (ex reserve a*
487 *translation file for proofreading, mark a translation as being ready to*
488 *be included in GNOME).*

489 *Committers are people with rights to make changes to the GNOME*
490 *translations that will be release. Unless a translations is not commit-*
491 *ted by a committer, it will only remain visible in the web interface,*
492 *as an attached PO file. Committers have access to all actions of a*
493 *reviewer with the addition of marking a PO file as committed and*
494 *archiving a discussion for new suggestions.*

495 The GNOME work-flow is characterized by everybody being able to suggest
496 translations, by having a big body of people who can review those and by
497 tightly controlling who can actually commit to the repositories. The possibility
498 of reserving translations also minimize the chances of wasting time translating
499 the same strings twice.

500 A very popular tool in the GNOME community of translators is the tool
501 Poedit⁴⁰, though the work-flow does not encourage a specific tool for the
502 translations themselves and GNOME translators do use several tools depending
503 on their personal preferences.

504 This graph illustrates their work-flow:

⁴⁰<http://www.poedit.net/>

