



Internationalization

# 1 Contents

2	Internationalization . . . . .	2
3	Text input . . . . .	2
4	Text display . . . . .	3
5	UI layout . . . . .	7
6	Localization . . . . .	8
7	Translation . . . . .	8
8	Testing . . . . .	12
9	Other locale configuration . . . . .	13
10	Distribution . . . . .	13
11	Runtime switching of locale . . . . .	15
12	Common pattern . . . . .	15
13	Application helper API . . . . .	16
14	Localization in GNOME . . . . .	16

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

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

22 All this will be done with the same tools used by GNOME and we do not antic-  
23 ipate any new development in the middleware itself, though UI components in  
24 the Apertis shell and applications will have to be developed with international-  
25 ization in mind, as explained in this document.

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

## 28 Internationalization

### 29 Text input

30 Some writing systems will require special software support for entering text, the  
31 component that provides this support for an specific writing system is called  
32 input method. There is a framework for input methods called [IBus](#)<sup>2</sup> that is  
33 the most common way of providing input methods for the different writing  
34 systems. Several input methods based on IBus are available in Ubuntu, and it  
35 is very unlikely that any needs will not be covered by them. An older, but more  
36 broadly-supported, input method framework is [SCIM](#)<sup>3</sup> and an even older one is  
37 [XIM](#)<sup>4</sup>.

---

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

<sup>2</sup>[http://en.wikipedia.org/wiki/Intelligent\\_Input\\_Bus](http://en.wikipedia.org/wiki/Intelligent_Input_Bus)

<sup>3</sup>[http://en.wikipedia.org/wiki/Smart\\_Common\\_Input\\_Method](http://en.wikipedia.org/wiki/Smart_Common_Input_Method)

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

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`<sup>5</sup> on top of `ClutterText`<sup>6</sup>, 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](#)<sup>7</sup>. As can be seen this implementation calls the following  
51 functions from the `GtkIMContext`<sup>8</sup> 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 reason-  
59 ably 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.

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

---

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

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

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

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

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)<sup>9</sup> 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  }
```

---

104 Note that `clutter_script_set_translation_domain()` or `textdomain()`<sup>10</sup> needs to

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

<sup>10</sup><http://linux.die.net/man/3/textdomain>

105 be called before translatable strings can be used in a ClutterScript file.

106 [gettext](#)<sup>11</sup> 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](#)<sup>12</sup>. However,  
109 in recent years, it has been superceded by [gettext](#)<sup>13</sup>. 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

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

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

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

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

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

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

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

---

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

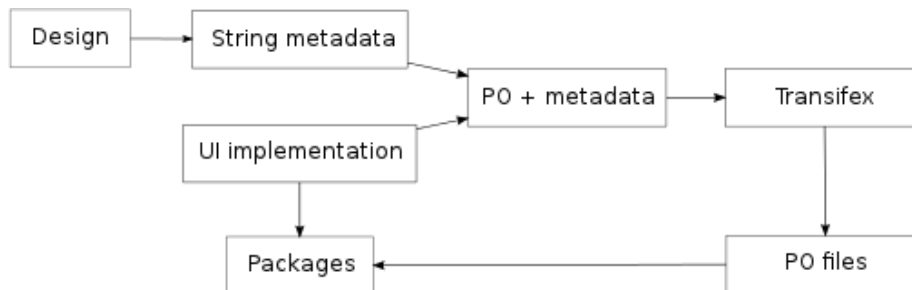
<sup>12</sup><https://launchpad.net/intltool/>

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

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

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

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



151

152 For better readability of the source code we recommend that the IDs chosen  
153 suggest the meaning of the string, such as *PARK\_ASSIST\_1*. Instead of hav-  
154 ing to specify whole font descriptions for each string to translate, Collabora  
155 recommends to use styles that expand to specific font descriptions.

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

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

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

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

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

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

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

### 183 **Consistency**

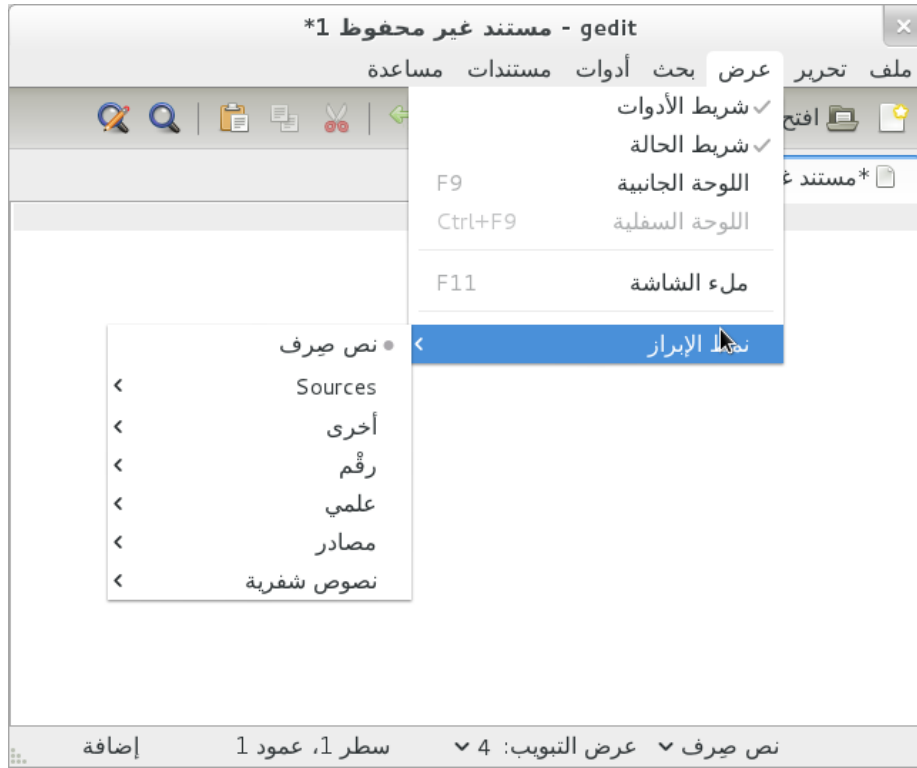
184 Translation management systems offer tools to increase the consistency of the  
185 translations, so the same words are used to explain the same concept. One of  
186 the tools that Transifex offers is a search feature that allows to quickly check  
187 how a word has been translated in other instances.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

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

### 192 **UI layout**

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

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



201

## 202 Localization

### 203 Translation

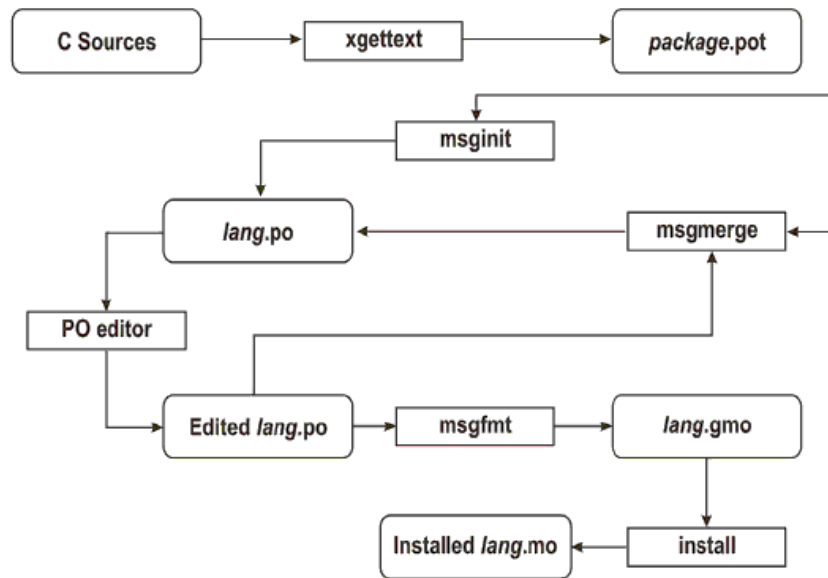
#### 204 GNU gettext

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

212 This diagram illustrates the [workflow](#)<sup>14</sup> when using GNU gettext to translate  
 213 text in an application written in C:

<sup>14</sup>[http://upload.wikimedia.org/wikipedia/commons/0/05/GNU\\_gettext\\_process.png](http://upload.wikimedia.org/wikipedia/commons/0/05/GNU_gettext_process.png)






---

### GNU gettext Working Process

214

215 From time to time, it is needed to extract new translatable strings from the  
 216 source code and update the files that are used by translators. The extraction  
 217 itself is performed by the tool `xgettext`<sup>15</sup>, which generates a new POT file con-  
 218 taining all the translatable strings plus their locations in the source code and  
 219 any additional context.

220 These are the `programming languages`<sup>16</sup> supported by GNU gettext: C, C++,  
 221 ObjectiveC, PO, Python, Lisp, EmacsLisp, librep, Scheme, Smalltalk, Java,  
 222 JavaProperties, C#, awk, YCP, Tcl, Perl, PHP, GCC-source, NXStringTable,  
 223 RST and Glade.

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

<sup>15</sup>[http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html\\_node/xgettext-Invocation.html](http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html_node/xgettext-Invocation.html)

<sup>16</sup>[http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html\\_node/xgettext-Invocation.html#index-supported-languages\\_002c-\\_0040cod](http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html_node/xgettext-Invocation.html#index-supported-languages_002c-_0040cod)

<sup>17</sup>[http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html\\_node/msgmerge-Invocation.html](http://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html_node/msgmerge-Invocation.html)

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

## 235 Translation management

236 Though these file generation steps can be executed manually with command  
237 line tools and translators can work directly on the `.po` files with any text editor,  
238 there are more high-level tools that aim to manage the whole translation process.  
239 Next we briefly mention the ones most commonly used in FOSS projects.

240 [Pootle](#)<sup>18</sup>, [Transifex](#)<sup>19</sup> and Launchpad [Rosetta](#)<sup>20</sup> are tools which provide conve-  
241 nient UIs for translating strings. They also streamline the process of translating  
242 strings from new `.pot` versions and offer ways to transfer the resulting `.po` files  
243 to source code repositories.

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

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

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

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

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

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

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

---

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

<sup>19</sup><https://www.transifex.com/>

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

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

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

## 268 **Transifex**

269 In this section we discuss in more details some aspects of Transifex. For an  
270 overview on other features of Transifex, please see the documentation for [man-](#)  
271 [agement](#)<sup>22</sup> and [translation](#)<sup>23</sup>.

## 272 **Deployment options**

273 Transifex is available as a hosted web service in <http://www.transifex.com><sup>24</sup> and  
274 there are

275 several [pricing options](#)<sup>25</sup> depending on the project size, features and level of  
276 technical support desired.

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

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

- 285 • Translation memory
- 286 • Glossary
- 287 • Improved collaboration between translators
- 288 • Improved UI theme

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

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

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

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

---

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

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

<sup>24</sup><http://www.transifex.com/>

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

<sup>26</sup><http://transifex.com/>

## 300 **Maintenance**

301 Most maintenance is performed through the web interface, by registered users  
302 of the web service with the appropriate level of access. This includes setting  
303 up users, teams, languages and projects. Less frequent tasks such as instance  
304 configuration, software updates, performance tuning and set up of automatic  
305 jobs are performed by the administrator of the server hosting the service.

## 306 **Translation memory**

307 Transifex will provide suggestions when translating a string based on existing  
308 [translations](#)<sup>27</sup> in the current module or in other modules that were configured to  
309 share their translation [memory](#)<sup>28</sup>. This memory can also be used to pre-populate  
310 translations for a new module based on other modules' [translations](#)<sup>29</sup>.

## 311 **Glossary**

312 Each project has a series of terms that are very important to translate con-  
313 sistentlly or that can have several different possible translations with slightly  
314 different meanings. To help with this, Transifex provides a [glossary](#)<sup>30</sup> that will  
315 assist translators in these cases.

## 316 **POT merging**

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

320 Once the PO files have been updated, the tool `tx` (command-line transifex client)  
321 can be used to submit the changes to the server, this merge happening as [fol-](#)  
322 [lows](#)<sup>31</sup>:

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

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

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

331 Note that this process can be automated.

---

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

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

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

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

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

### 332 **Automatic length check**

333 Transifex’s database model will have to be updated to store additional metadata  
334 about each string such as the font description and the available size in pixels.  
335 The web application could then check how many pixels the entered string would  
336 take in the UI, using Pango and [Fontconfig](#)<sup>32</sup>. For better accuracy, the exact  
337 fonts that will be used in the UI should be used for this computation.

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

### 340 **Testing**

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

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

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

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

366 If, despite automatically warning the translator when a translation will not fit  
367 in the UI, some strings are too long, the UI widget that displays the string could  
368 ellipsize it to indicate that the displayed text isn’t complete. If this occurred  
369 in a debug build, a run-time warning could be also emitted. These warnings

---

<sup>32</sup><http://fontconfig.org/>

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

370 would be logged only once a translated string has been displayed in the UI and  
371 wouldn't apply to text coming from an external input.

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

## 374 Other locale configuration

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

## 379 Distribution

380 There are three main ways of packaging translations:

- 381 • package all the MO files (compiled PO files) along the rest of the files for  
382 a single component (for example gnome-shell in Ubuntu).
- 383 • package the MO files for a single component (usually a big one such as  
384 LibreOffice or KDE) and a specific language in a separate package (for  
385 example, [firefox-locale-de](#)<sup>34</sup> in Ubuntu).
- 386 • package several MO files corresponding to several components for one  
387 language (for example language-pack-cs-base in Ubuntu).

388 Our recommendation at this stage is to have:

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

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

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

---

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

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

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

Package name	Contents	Package size	Installed size
language-pack-de-base	MO files for core packages <sup>1</sup>	2,497 kB	8,432 kB
firefox-locale-de	German translation for Firefox <sup>2</sup>	233 kB	453 kB
libreoffice-l10n-de	Resource files with translations, and templates <sup>3</sup>	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 <sup>4</sup>	388 kB	1,496 kB

408 The *language-support-fonts-ja* package is a virtual one that brings the following  
409 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

410 Modern distributions will bring all those fonts for Japanese-enabled installa-  
411 tions, but depending on the commercial requirements, a system could make  
412 with just a subset. Similarly, other locales will require a set of fonts for prop-  
413 erly rendering text in the same way as users in specific markets expect. In order  
414 to recommend specific font files, knowledge on the requirements are needed.

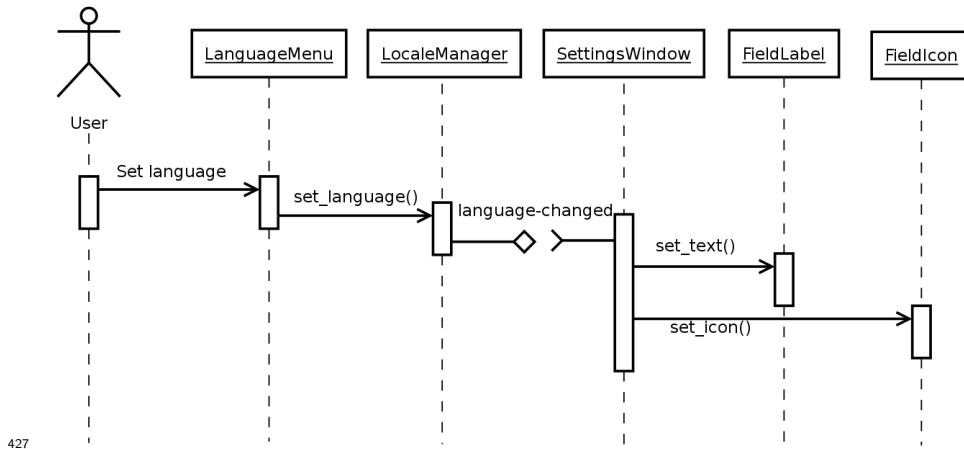
## 415 Runtime switching of locale

### 416 Common pattern

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

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

<sup>36</sup>[https://wiki.qt.io/How\\_to\\_create\\_a\\_multi\\_language\\_application](https://wiki.qt.io/How_to_create_a_multi_language_application)



427

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

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

435 Since `systemd` version 30, the `systemd-localed` service<sup>37</sup> has been provided as  
 436 a standard D-Bus API (`org.freedesktop.locale1`) for managing the system's  
 437 locale, including being notified when it is changed, getting its current value,  
 438 and setting a new value. This should be used in combination with the  
 439 `org.gnome.system.locale` GSettings schema, which stores the *user's* locale pref-  
 440 erences. We suggest that the `LocaleManager` from the diagram is implemented  
 441 to query `org.gnome.system.locale` and returns the value of its `region` setting if  
 442 set. If not set, the user is using the default system locale, which `LocaleManager`  
 443 should query from `org.freedesktop.locale1`.

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

447 `systemd-localed` uses `polkit`<sup>38</sup> to authorise changes to the system locale, so ven-  
 448 dors would need to write a policy which determines which applications are per-  
 449 mitted to change the system locale, and which are allowed to query it. The  
 450 default should be that only the system preferences application is allowed to  
 451 change the locale; and all applications are allowed to query it (and be notified  
 452 of changes to the locale).

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

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



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

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

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

### 461 Application helper API

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

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

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

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

### 474 Localization in GNOME

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

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

484 *Translators contains persons helping with GNOME translations into*  
485 *a specific language, who added themselves to the translation team.*  
486 *Translators could add comment to a specific PO file translation, could*  
487 *reserve it for translations and could suggest new translations by up-*  
488 *load a new PO file. The suggested translations will be reviewed by*  
489 *other team members.*

---

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

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

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

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

507 A very popular tool in the GNOME community of translators is the tool  
508 [Poedit](http://www.poedit.net/)<sup>40</sup>, though the work-flow does not encourage a specific tool for the  
509 translations themselves and GNOME translators do use several tools depending  
510 on their personal preferences.

511 This graph illustrates their work-flow:

---

<sup>40</sup><http://www.poedit.net/>

