Qt Linguist Manual: Text ID Based Translations

The text ID translation mechanism is an "industrial strength" system for internationalization and localization. Each text in the application is assigned a unique identifier (text ID) and these identifiers are used directly in the source code in place of the plain texts. This requires a little more work for the user interface developers but makes it much easier to manage large numbers of translated texts.

Note: You must use only plain-text-based or only text-ID-based functions in one application. If you mix them, you will end up with an incomplete set of texts to be translated.

Internationalizing With Text IDs

When using text IDs instead of plain text, the general method of internationalizing an application is the same but the details are a bit different:

1. The functions and macros for the text-ID-based translation system are different to the plain-text system. You use the `qsTrId()` function instead of qsTr(), the `QT_TRID_NOOP()` macro instead of QT_TR_NOOP(), and `QT_TRID_N_NOOP()` macro instead of QT_TR_N_NOOP()).
2. Use text IDs as user interface strings rather than plain text strings. For example, `qsTrId("id-back-not-front")`
3. You cannot specify a context parameter with a text ID. If there are identically spelled words with different meanings, these need separate text IDs. For example, `qsTrId("id-back-backstep")` will differentiate the back-step "Back" from the back-of-the-object "Back".
4. The "Engineering English" text that you see in the user interface for development builds is indicated with a `//%` comment. If you do not include this, the text ID will be shown in the user interface. This is especially important when you have texts with parameters. The `//%` comment needs to include the parameters indicators in the string. For example, `//% "Number of files: %1"`
5. The `//:` comments that provide extra information to the translator are optional in the plain-text system. However, with the text-ID-based system, this extra information becomes essential because without it you only have the text ID and the translator might not be able to make a sensible translation from that without further context. You can use long descriptive text IDs and no comments, but comments are often easier to understand.

The side-by-side code snippets below show a comparison of text-ID -based and plain-text-based translations:

text-ID-based	plain-text-based

```
Text {
    id: backTxt;
    //: The back of the object, not the front
    //% "Back"
    //~ Context Not related to back-stepping
    text: qsTrId("id-back-not-front");
}
```
	```
	Text {
	    id: backTxt;
	    //: The back of the object, not the front
	    //~ Context Not related to back-stepping
	    text: qsTr("Back","Not front")
	}
	```

Localizing With Text IDs

Localizing with text IDs follows much the same process as for plain text.

The `lupdate` tool is used the same way and translations are made into the .ts files:

```
lupdate <myapp>.pro
```

Note that the source values in the translation files will be text IDs rather than plain text. This means you need very descriptive text IDs, or good additional comments, or both to ensure that the translator makes a correct translation.

The example text-ID-based user interface text from above results in the following content in the .ts file:

```
<message id="id-back-not-front">
    <source>Back</source>
    <extracomment>The back of the object, not the front</extracomment>
    <translation type="unfinished"></translation>
    <extra-Context>Not related to back-stepping</extra-Context>
</message>
```

When using `lrelease`, you need to specify that the keys for translated texts are based on text IDs, rather than plain texts. If strings in the code are specified with `qsTr()` there is no "id" attribute set so they are ignored by `lrelease`.

This command produces all the compiled translation .qm files for your application:

```
lrelease -idbased <myapp>.pro
```

However, if there is no translation available for a given text (which is generally the case until late in development), the text ID will be shown in the user interface rather than a proper text. In order to make the application more usable for testing, you can make `lrelease` use the "Engineering English" source text (from the `//%` comments) as the translated text and mark it with some indicator so you can see texts that are not yet translated.

For example, this command builds the .qm files and puts a "!" in front of the untranslated texts:

```
lrelease -idbased -markuntranslated ! <myapp>.pro
```

Advanced Usage

For projects that target a large number of locales, you can remove the TRANSLATIONS info from the .pro file and, instead, manage the translations with a separate script. The script can call lrelease and lupdate for each of the desired targets.

The updates could be scripted something like this:

```
lupdate -recursive <project-dir> -ts <project-dir>/i18n/myapp-text_en_GB.ts
lupdate -recursive <project-dir> -ts <project-dir>/i18n/myapp-text_en_US.ts
...
```

```
lrelease -idbased <project-dir>/i18n/myapp-text_en_GB.ts
lrelease -idbased <project-dir>/i18n/myapp-text_en_US.ts
...
```

< Qt Linguist Manual: TS File Format

The Qt Company

Contact Us

Company

About Us
Investors
Newsroom
Careers
Office Locations

Licensing

Terms & Conditions
Open Source
FAQ

Support

Support Services
Professional Services
Partners
Training

For Customers

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum
Wiki
Downloads
Marketplace

© 2022 The Qt Company

Feedback Sign In