Localizing Qt application on Symbian

This article explains how to use localized resources in a Qt C++ app. This is valid for Qt 4.7 onwards and Nokia Belle.

Introduction

Localization of Qt applications on Symbian are somewhat different than native Symbian localization.

Details

Qt uses translation mechanism for localization. The translation files (with extension .ts) for each language should be available for localization support. However for using in application, .ts files need to be converted to .qm. Two tools are available for the conversion: lupdate and lrelease. These tools can process qmake project files, or operate directly on the file system. lrelease is part of Qt's Linguist tool chain. It can be used as a stand-alone tool to convert XML-based translations files in the TS format into the 'compiled' QM format used by QTranslator objects.

```
lrelease Usage:
     lrelease [options] project-file
     lrelease [options] ts-files [-qm qm-file]

 Options:
     -help  Display this information and exit
     -idbased
            Use IDs instead of source strings for message keying
     -compress
            Compress the QM files
     -nounfinished
            Do not include unfinished translations
     -removeidentical
            If the translated text is the same as
            the source text, do not include the message
     -markuntranslated <prefix>
            If a message has no real translation, use the source text
            prefixed with the given string instead
     -silent
            Do not explain what is being done
     -version
            Display the version of lrelease and exit

lupdate Usage:
     lupdate [options] [project-file]
     lupdate [options] [source-file|path]... -ts ts-files
 Options:
     -help  Display this information and exit.
     -noobsolete
            Drop all obsolete strings.
     -extensions <ext>[,<ext>]...
            Process files with the given extensions only.
            The extension list must be separated with commas, not with whitespace.
            Default: 'ui,c,c++,cc,cpp,cxx,ch,h,h++,hh,hpp,hxx'.
     -pluralonly
            Only include plural form messages.
     -silent
            Do not explain what is being done.
     -version
            Display the version of lupdate and exit.
```

The easiest method to use lupdate and lrelease is by specifying a .pro Qt project file. There must be an entry in the TRANSLATIONS section of the project file for each language that is additional to the native language. A typical entry looks like this:

```
 TRANSLATIONS = myapp_fr.ts \
                myapp_nl.ts
  CODECFORTR = ISO-8859-5
```

Using a locale within the translation file name is useful for determining which language to load at runtime. The .ts files will be converted to .qm with same file name.

Now the .qm files should be loaded in the application. Typically translation file is loaded in main() and are used from anywhere in the project. The sample loading code is like the following:

```
int main(int argc, char *argv[])
{
    QCoreApplication app(argc, argv);

    //adding localization
    QTranslator* translator = new QTranslator(0);
    #ifdef __SERIES60_
    QString translatorFileName = QLatin1String("myapp_");
    translatorFileName += QLocale::system().name() + ".qm";
    QString dirName = QLibraryInfo::location(QLibraryInfo::TranslationsPath);
    if (!translator->load(translatorFileName, dirName))
        qDebug() << "Unable to load localization file.";
    #else
    if(!translator->load("language/myapp.qm"))
    qDebug() << "Unable to load localization file.";
    #endif
    else
        qDebug() << "Successfully loaded localization file.";
    app.installTranslator(translator);  //translator installed
...
...
```


(C) Copyright Nokia 2013. All rights reserved.

In the above code, all .qm files are stored under "./language" directory. Once the translator is installed, the translation for a localized string can be retrieved like the following code:

```
QString usedString = qtTrId("qtn_sample_string");
```

Typically "qtn_sample_string" will be defined in each localization (.ts) file like this:

```
<?xml version="1.0" encoding="utf-8"?>
 <!DOCTYPE TS>
 <TS version="3.0" sourcelanguage="en_GB">
 <context>
 <message numerus="no" id="qtn_sample_string">
 <comment>A sample string. Some description goes here.</comment>
 <source>Sample</source>
 <translation variants="no" type="unfinished"></translation>
 <extra-loc-layout_id>Temporary</extra-loc-layout_id>
 <extra-loc-viewid>MyApp</extra-loc-viewid>
 <extra-loc-positionid>Popup_info_1</extra-loc-positionid>
 <extra-loc-feature>MyApp</extra-loc-feature>
 <extra-loc-blank>False</extra-loc-blank>
 </message>
 </context>
 </TS>
```

And a corresponding arabic translation file looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<TS version="1.0" sourcelanguage="en" language="ar">
<context>
<name>QtApplicationCaptions</name>
<message numerus="no" id="qtn_sample_string">
<source>Source</source>
<translation variants="no">عينة</translation>
</message>
</context>
</TS>
```

The search logic for loading translator file present in the `main()` code has looped logic. i.e., the call `translator->load(translatorFileName, dirName)` will first try to find the file with the locale name added at the end (see above code). So if EN_GB is the current selected locale in the device, it will search for "myapp_EN_GB.qm" in the translation path provided. If not found, it will search for "myapp_EN.qm" and if still not found, it will search for "myapp.qm". So translation files' naming should follow such intelligent naming convention.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.