Вывод русскоязычных сообщений в Qt4. Материалы семинаров.

Самарев Роман Станиславович, samarev [ ] acm.org
канд. техн. наук, доцент каф. «Компьютерные системы и сети», МГТУ им. Н.Э.Баумана, 2013.

При написании интерактивных программ очевидна необходимость вывода сообщений на языке, понятном пользователю. Однако проблема заключается в том, что программы на C и C++ пишутся с использованием основного набора символов, который не включает кириллический алфавит. Несмотря на то, что эти языки позволяют использовать в строках символы, отличные от латиницы, нет единого способа указания кодировки файла, а она может быть любой: CP1251, CP866, UTF-8, KOI8-R…. Компилятор C++, который получает на вход текст программы, не в состоянии определить или изменить кодировку строк, если она не соответствует кодировке операционной системы. Подобные проблемы возникают в тех случаях, когда программа предназначена для работы в нескольких операционных системах, но реализуется в одной из них.

При работе с библиотекой Qt существует несколько способов корректного преобразования строк, содержащих символы в национальных алфавитах. Напомним, что строки в Qt представлены в кодировке UNICODE, поэтому при выводе в виджетах они однозначно будут корректно отображены. Преобразование из UNICODE в кодировку операционной системы, если оно необходимо, Qt выполнит автоматически. Следовательно, проблема вывода строк разбивается на 3 случая:

- строки текста программы содержат символы не основного алфавита;
- текст программы не содержат символы не основного алфавита;
- текст программы имеет кодировку UTF-8 или UTF-16.

В виду того, что для C++ кодировка UNICODE является инородной, потенциально могут возникнуть ошибки с использованием типов char и wchat_t, а также проблемы с различными компиляторами, которые не смогут самостоятельно определить кодировку, поэтому данный вариант кодирования строк не следует использовать. Рассмотрим два оставшихся варианта.

*Использование национальных символов в строках char**

Простейший случай – программы, ориентированные на пользователей конкретного региона. В этом случае нет необходимости предусматривать вывод сообщений на нескольких языках, а строковые литералы (то есть строки-константы) могут быть написаны с использованием соответствующей кодировки символов.
В этом случае программист всегда знает, какая кодировка использована при написании программы, поэтому преобразование кодировки в UNICODE для преобразования в строки QString может быть заложено на этапе кодирования.

Для преобразования кодировок в Qt4 существует класс `QTextCodec`. Этот класс имеет методы для преобразования строк из указанной кодировки в UNICODE и обратно, а также методы, устанавливающие кодировку для строк типа char*, например:
void QTextCodec::setCodecForCStrings (QTextCodec * codec).

Для начала работы необходимо получить кодек для указанной кодировки. По-умолчанию при наборе текста программы в MS Visual Studio используется кодировка CP1251, которая

традиционна для MS Windows. Поэтому указана именно эта кодировка. Заметим, что в ряде специальных сборок ОС Linux может встретиться кодировка KOI8-R.

```
#define APP_STRING_CODECNAME    "CP1251"
QTextCodec  *pCodec = QTextCodec::codecForName(APP_STRING_CODECNAME);
```

После этого получаем возможность использования функций преобразования в UNICODE и из UNICODE, которые удобно оформить в виде define:

```
#define RUS(a) pCodec->toUnicode(a)
#define U_TO_RUS_DECLARE      QByteArray ba_U_TO_RUS;
#define U_TO_RUS(a) (ba_U_TO_RUS = m_pCodec->fromUnicode(a)).data()
```

В том случае, когда все строки с которыми будет работать программа имеют одну единственную кодировку, можно воспользоваться методами, устанавливающими кодировку по-умолчанию для всех строк, например QTextCodec::setCodecForCStrings. В прочих случаях лучше воспользоваться ранее определенными макросами RUS и U_TO_RUS:

```
ui->label->setText(
     RUS("Некоторый текст, введенный в кодировке, соответствующей cpp-файлу"));
```

Поскольку результатом выполнения метода `toUnicode()` будет объект QString, то над ним возможно выполнение любых операций (склейка, поиск и пр.), возможных для QString.

Отметим также, что в программе, использующей несколько форм, целесообразно создание глобальной переменной-кодека, который будет инициализироваться до создания первой формы.

Указанные способ преобразования строк также применим при работе с текстовыми файлами в разных кодировках.

Интернационализация интерфейса приложений

Рассмотренный выше способ преобразования позволяет непосредственно в тексте программы использовать строковые литералы, содержащие национальные символы, однако этот способ не отвечает современной потребности в реализации программ, способных отображать интерфейс на различных языках. Основной принцип интернационализации приложений заключается в том, чтобы разделить код программы и отображаемый текст. За код программы отвечает программист, а за отображаемый текст – носитель языка. При этом программист получает возможность реализовывать программные продукты на тех языках, которые не знает.

Для реализации многоязычного интерфейса в Qt4 существуют специальные средства: для всех классов-потомков QObject – метод tr(), для всех остальных случаев макросы QT_TR_NOOP() или QT_TRANSLATE_NOOP() или метод QCoreApplication::translate().

Основной принцип заключается в том, что текст программы не должен содержать никаких символов, отличных от символов основной латинской группы. Надписям, включенным в текст программы, должен соответствовать текст, расположенный во внешних ts-файлах переводов, имеющих кодировку UNICODE. Рассмотрим последовательность действий.

Шаг 1. Необходимо добавить в файл проекта указание на файлы перевода. Для этого в pro-файл (здесь будет использоваться linguist.pro) следует добавить имена файлов для всех языков, которые необходимо реализовать:

```
TRANSLATIONS += translations/ru.ts \
                translations/en.ts
```

Обычно файлы переводов размещают в директории translations и её необходимо создать.

Обратите внимание на то, что указаны переводы как для русского, так и для английского языков. Несмотря на то, что строки на английском языке могут быть непосредственно вставлены в код программы, такой подход позволяет возложить задачу качественно перевода на носителя языка, а не заботиться об этом на этапе написания кода программы. То есть в программе могут быть вставлены абсолютно любые строки, например мнемонические имена типа SELECT_FILE, ORDER_NUM, а файлы перевода уже будут содержать полноценный текст на целевых языках.

Шаг 2. Необходимо сгенерировать шаблоны файлов переводов.

Для генерации или обновления файлов перевода по уже имеющимся файлам исходных текстов программы необходимо воспользоваться утилитой lupdate из комплекта Qt. Утилита lupdate может автоматически обработать файл проекта, поэтому для обновления файлов переводов запустите в консоли команду:

```
lupdate linguist.pro
```

Утилита lupdate позволяет также вручную запускать обновление файла перевода для конкретного файла: `lupdate ..\GeneratedFiles\ui_mainwindow.h -ts ru.ts`

Принцип работы lupdate заключается в том, что утилита ищет в указанных исходных файлах программы вызовы методов tr() или translate(), после чего формирует соответствующие записи в файле. Файл перевода, или ts-файл должен быть создан для каждого языка интерфейса, является xml-файлом и, для рассматриваемого примера, имеет следующий вид:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE TS>
<TS version="2.0">
<context>
    <name>MainWindow</name>
    <message>
        <location filename="../mainwindow.ui" line="14"/>
        <source>MainWindow</source>
        <translation type="unfinished"></translation>
    </message>
    <message>
        <location filename="../mainwindow.ui" line="21"/>
        <source>Label1</source>
        <translation type="unfinished"></translation>
    </message>
    <message>
        <location filename="../mainwindow.ui" line="31"/>
        <source>Label2</source>
        <translation type="unfinished"></translation>
    </message>
    <message>
        <location filename="../mainwindow.ui" line="46"/>
        <source>Button1</source>
        <translation type="unfinished"></translation>
    </message>
    <message>
        <location filename="../mainwindow.ui" line="53"/>
        <source>Button2</source>
        <translation type="unfinished"></translation>
    </message>
</context>
</TS>
```

Исходный файл `mainwindow.ui` - это форма, подготовленная в Qt Designer. Этот файл в процессе компиляции программы обрабатывается компилятором uic, после чего формируется файл `ui_mainwindow.h`. В этом сгенерированном файле будет включена следующая функция:

```
void retranslateUi(QMainWindow *MainWindow)
{
    MainWindow->setWindowTitle(QApplication::translate("MainWindow",
       "MainWindow", 0, QApplication::UnicodeUTF8));
    label->setText(QApplication::translate("MainWindow",
      "Label1", 0, QApplication::UnicodeUTF8));
    label_2->setText(QApplication::translate("MainWindow",
      "Label2", 0, QApplication::UnicodeUTF8));
    pushButton->setText(QApplication::translate("MainWindow",
      "Button1", 0, QApplication::UnicodeUTF8));
    pushButton_2->setText(QApplication::translate("MainWindow",
      "Button2", 0, QApplication::UnicodeUTF8));
} // retranslateUi
```

Именно эти вызовы `translate()` осуществят подстановку текста перевода вместо исходных строк.

Отметим, что в минимальном случае при вставке текста, подлежащего перекодированию, достаточно указать единственный параметр метода tr – строковый литерал для перевода.

Утилита lupdate может выделять строки как из cpp-файлов, так и из файлов форм ui, причем при указании в качестве параметра pro-файла результат обработки всех файлов исходных текстов будет помещен в общий ts-файл.

Шаг 3. Подготовка перевода.

Обратите внимание на то, что ts-файл имеет кодировку UTF-8, имеет явное указание языка, для которого сделан перевод, а также имеет разметку переводов, позволяющую однозначно идентифицировать расположение исходных строк.

Этот файл может быть отредактирован вручную или с помощью специального средства Qt Linguist, которое позволяет удобно редактировать перевод в контексте форм, управляющих элементов, а также выполнить проверку знаков препинания, комбинаций клавиш-акселераторов, совпадение фраз и совпадение маркеров.

В зависимости от того, что являлось исходным файлом для формирования файла перевода, Qt Linguist будет подсвечивать для выбранной переводимой константы либо положение в исходном коде (см. рисунок 1), либо положение элемента на форме (см. рисунок 2).

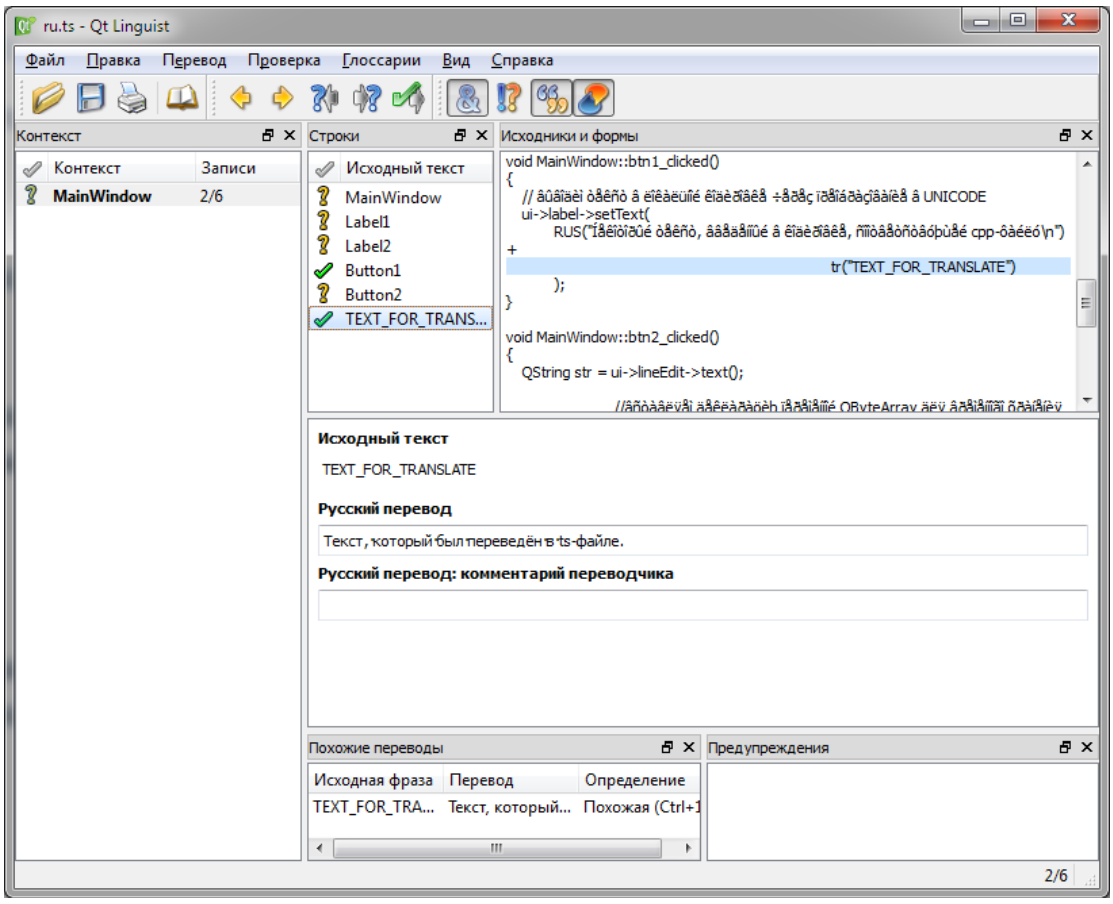

Рисунок 1 Подсветка исходного кода в Qt Linguist.

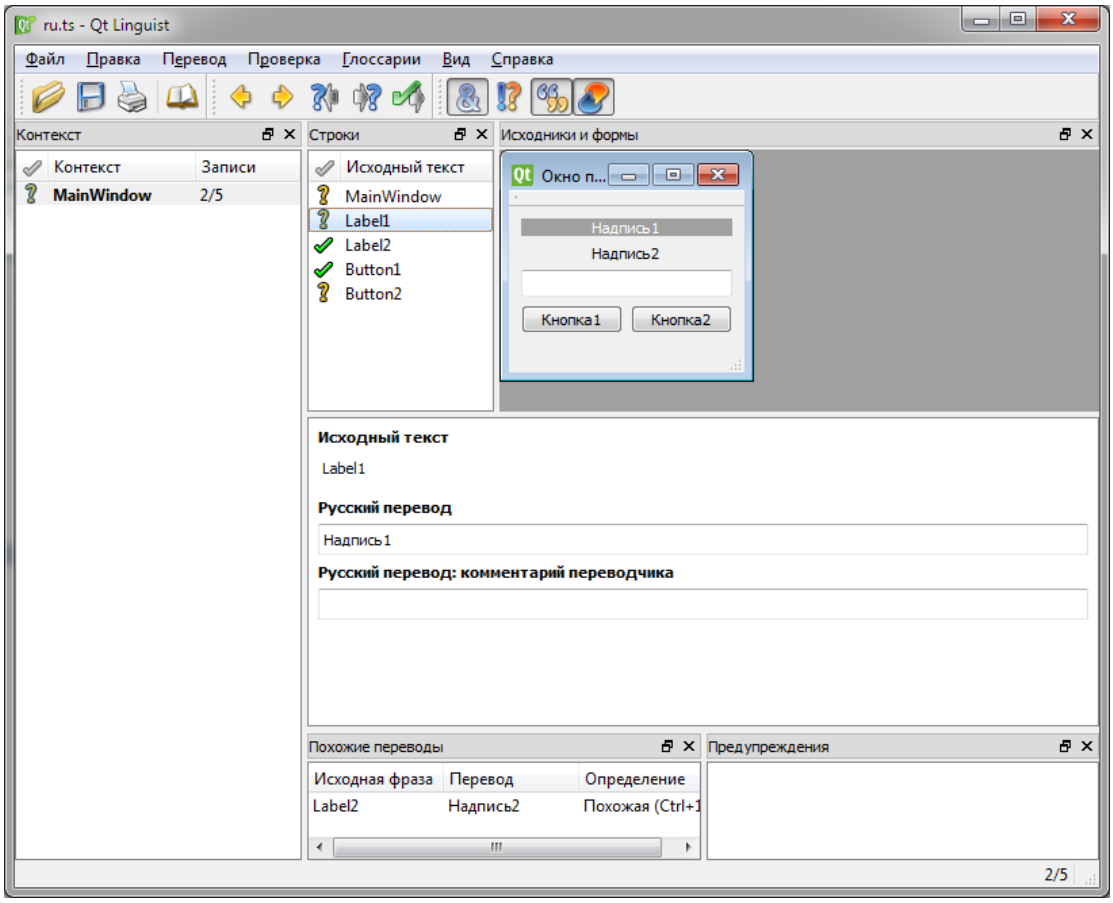

Рисунок 2 Подсветка элемента формы в Qt Linguist.

Обратите внимание на то, что для каждой переводимой строки может быть установлен статус завершенности (знак √) или не завершенности (вопросительный знак) перевода.

Шаг 4. После окончания редактирования необходимо сформировать сжатое представление файлов перевода, то есть qm-файлы. Это можно сделать утилитой lrelease, указав в качестве параметра pro-файл или конкретный ts-файл:

```
lrelease linguist.pro
```

В результате в директории translates будут сгенерированы файл ru.qm и en.qm.

Шаг 5. Функции перевода будут использоваться в программе, однако файл перевода не будет подключен автоматически. Для этого необходимо подключить его следующим образом:

```
    QTranslator translator;
    if( translator.load("translations/ru.qm") )
      qApp->installTranslator(&translator);
```

Обратите внимание на то, что файл перевода должен находиться в директории, которая указана при вызове метода `translator.load`, причем **формат пути – UNIX**! Такой способ загрузки перевода позволяет добавлять файлы перевода без необходимости пересборки программы. Однако при желании можно интегрировать файлы перевода в состав единственного исполняемого файла приложения. Для этого существует механизм ресурсов.

Шаг 6. Добавление перевода в ресурсы. Для подключения ресурсов необходимо добавить в pro-файл соответствующую строку:
RESOURCES += linguist.qrc

Файл ресурсов linguist.qrc имеет следующий вид:

```
<RCC>
  <qresource prefix="/" >
    <file>translations/ru.qm</file>
    <file>translations/en.qm</file>
  </qresource>
</RCC>
```

Подключаемые ресурсы являются файлами и указаны в элементах <file> по относительным путям к текущему расположению qrc-файла, причем обязательно в UNIX-формате. Также возможно добавление корневого префикса пути в тех случаях, когда необходимо логически разделить ресурсы.
Подключение файлов переводов в этом случае производится следующим образом:

```
int main(int argc, char *argv[])
{
    Q_INIT_RESOURCE(linguist);
    QApplication a(argc, argv);

    QTranslator translator;
    if( translator.load(":/translations/ru.qm") )
      qApp->installTranslator(&translator);

    MainWindow w;
    w.show();
    return a.exec();
}
```

Здесь следует обратить внимание на строку инициализации ресурсов Q_INIT_RESOURCE(linguist) и формат имени файла при вызове `translator.load`. Путь к локальным, то есть встроенным в исполняемый файл ресурсам начинается с символа ‘:’. Далее следует префикс, указанный в файле ресурсов <qresource prefix="/" >. И лишь после этого следует имя файла конкретного ресурса. В указанном примере подключается только русский язык, однако в общем случае язык перевода подключается на основании текущих настроек операционной системы (соответствие системной локали) или в соответствии с установленными настройками.

Если выбор языка отображения производится во время работы программы, необходимо предусмотреть обновление по событию `QEvent::LanguageChange`. Для этого необходимо переопределить в главных виджетах (например формы) виртуальный метод `changeEvent`. Пример из документации «Internationalization with Qt»:

```
void MyWidget::changeEvent(QEvent *event)
 {
     if (e->type() == QEvent::LanguageChange) {
         titleLabel->setText(tr("Document Title"));
         ...
         okPushButton->setText(tr("&OK"));
     } else
         QWidget::changeEvent(event);
 }
```

Литература

Internationalization with Qt - [эл. рес. - http://qt-project.org/doc/qt-4.8/internationalization.html]

Приложение. Исходный код программы для иллюстрации перевода

В качестве примера приведем приложение с формой, на которой присутствует две надписи и две кнопки (рисунок 3). Нажатие кнопки 1 приводит к выводу заложенного в коде программы текста. Нажатие кнопки 2 приводит к преобразованию текста, который введен в поле для ввода, а именно код каждого символа будет увеличен на 1, причём преобразование будет проведено в кодировке cp1251.

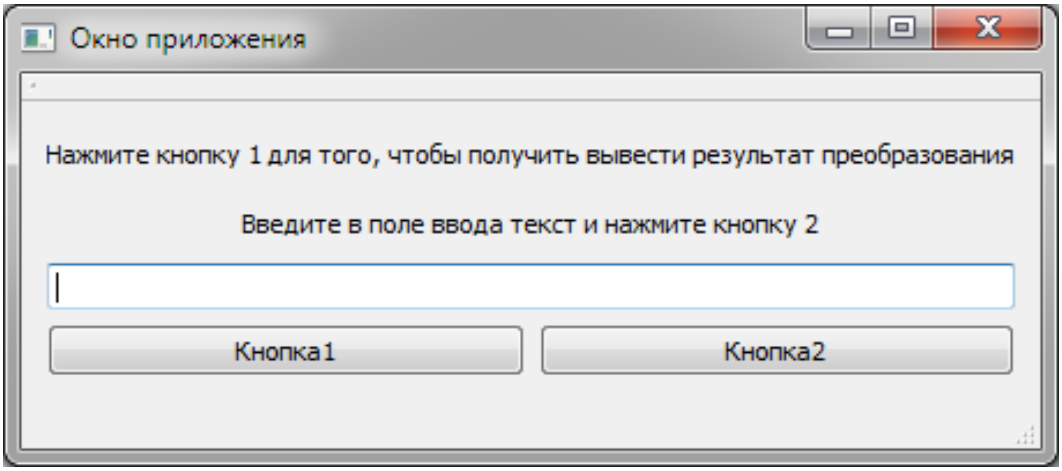

Рисунок 3 Форма тестового приложения.

Файл проекта linguist.pro

```
TARGET = linguist
TEMPLATE = app
SOURCES += main.cpp \
           mainwindow.cpp
HEADERS += mainwindow.h
FORMS += mainwindow.ui
RESOURCES += linguist.qrc
TRANSLATIONS += translations/ru.ts \
                translations/en.ts
```

Файл ресурсов linguist.qrc

```
<RCC>
    <qresource prefix="/" >
        <file>translations/ru.qm</file>
        <file>translations/en.qm</file>
    </qresource>
</RCC>
```

Файл main.cpp

```
#include <QtGui/QApplication>
#include <QTranslator>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    Q_INIT_RESOURCE(linguist);
    QApplication a(argc, argv);

    // Устанавливаем язык для преобразовния строк, использующих функцию tr()
    // или translate(). В данном примере они содержаться в файле ui_mainwindow.h
    // (генерируется по ui-файлу).
    // Внимание! Строки, которые будут преобразовываться через QTranslator
    // должны содержать только латинские символы. В противном случае будут
    // проблемы с написанием ts-файлов.
    // Файл ru.ts сгенерирован утилитой lupdate linuguist.pro и
    // отредактирован в QtLinguist
    // Файл ru.qm сформирован утилитой lrelease linuguist.pro
    QTranslator translator;
    if( translator.load(":/translations/ru.qm") )
      qApp->installTranslator(&translator);

    MainWindow w;
    w.show();
    return a.exec();
}
```

Файл mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>218</width>
    <height>162</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <widget class="QLabel" name="label">
      <property name="text">
       <string>Label1</string>
      </property>
      <property name="alignment">
       <set>Qt::AlignCenter</set>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QLabel" name="label_2">
      <property name="text">
       <string>Label2</string>
      </property>
      <property name="alignment">
       <set>Qt::AlignCenter</set>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QLineEdit" name="lineEdit"/>
    </item>
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout">
      <item>
       <widget class="QPushButton" name="pushButton">
        <property name="text">
         <string>Button1</string>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QPushButton" name="pushButton_2">
        <property name="text">
         <string>Button2</string>
        </property>
       </widget>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
  <widget class="QMenuBar" name="menuBar">
```

```
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>218</width>
     <height>19</height>
    </rect>
   </property>
  </widget>
  <widget class="QToolBar" name="mainToolBar">
   <attribute name="toolBarArea">
    <enum>TopToolBarArea</enum>
   </attribute>
   <attribute name="toolBarBreak">
    <bool>false</bool>
   </attribute>
  </widget>
  <widget class="QStatusBar" name="statusBar"/>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections>
  <connection>
   <sender>pushButton</sender>
   <signal>clicked()</signal>
   <receiver>MainWindow</receiver>
   <slot>btn1_clicked()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>57</x>
     <y>110</y>
    </hint>
    <hint type="destinationlabel">
     <x>5</x>
     <y>114</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>pushButton_2</sender>
   <signal>clicked()</signal>
   <receiver>MainWindow</receiver>
   <slot>btn2_clicked()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>139</x>
     <y>110</y>
    </hint>
    <hint type="destinationlabel">
     <x>159</x>
     <y>131</y>
    </hint>
   </hints>
  </connection>
 </connections>
 <slots>
  <slot>btn1_clicked()</slot>
  <slot>btn2_clicked()</slot>
 </slots>
</ui>
```

Файл mainwindow.cpp (кодировка CP1251)

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

// Указываем кодировку, в которой вводим надписи на русском языке
//#define APP_STRING_CODECNAME    "KOI8-R"
#define APP_STRING_CODECNAME    "CP1251"

// делаем макроопределения для преобразования в юникод и обратно
#define RUS(a) m_pCodec->toUnicode(a)

#define U_TO_RUS_DECLARE      QByteArray ba_U_TO_RUS;
#define U_TO_RUS(a) (ba_U_TO_RUS = m_pCodec->fromUnicode(a)).data()

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow)
{
    // создаем кодек для выбранной кодировки
    m_pCodec = QTextCodec::codecForName(APP_STRING_CODECNAME);

    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
      // ui необходимо удалить явно, поскольку класс не является Qt-классом.
    delete ui;
}

void MainWindow::btn1_clicked()
{
    // выводим текст в локальной кодировке через преобразование в UNICODE
    ui->label->setText(
            RUS("Некоторый текст, введенный в кодировке, соответствующей cpp-
файлу\n") +
                    tr("TEXT_TO_TRANSLATE")
            );
}

void MainWindow::btn2_clicked()
{
    QString str = ui->lineEdit->text();

      //вставляем декларацию переменной QByteArray для временного хранения строк
      U_TO_RUS_DECLARE;
      // переводим из UNICODE в нужную кодировку,
      // если нет возможности обработать в UNICODE
      char * str_c = U_TO_RUS(str);

      // какая-то обработка строки как (char*)
      for( size_t i=0; str_c[i]; i++ ){
            str_c[i] = str_c[i]+1;
      }

      // преобразуем текст обратно в UNICODE
      ui->label_2->setText( RUS(str_c) );
}
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QtGui/QMainWindow>
#include <QTextCodec>

namespace Ui
{
    class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    // Кодек для использования в форме
    QTextCodec * m_pCodec;

private slots:
    void btn1_clicked();
    void btn2_clicked();
};

#endif // MAINWINDOW_H
```

Файл перевода translations/ru.ts (кодировка UTF-8)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE TS>
<TS version="2.0" language="ru_RU">
<context>
    <name>MainWindow</name>
    <message>
        <location filename="../mainwindow.ui" line="14"/>
        <source>MainWindow</source>
        <translation type="unfinished">Окно приложения</translation>
    </message>
    <message>
        <location filename="../mainwindow.ui" line="21"/>
        <source>Label1</source>
        <translation type="unfinished">Нажмите кнопку 1 для того, чтобы вывести
результат преобразования.</translation>
    </message>
    <message>
        <location filename="../mainwindow.ui" line="31"/>
        <source>Label2</source>
        <translation type="unfinished">Введите в поле ввода текст и нажмите
кнопку 2.</translation>
    </message>
    <message>
        <location filename="../mainwindow.ui" line="46"/>
        <source>Button1</source>
        <translation>Кнопка1</translation>
    </message>
    <message>
        <location filename="../mainwindow.ui" line="53"/>
        <source>Button2</source>
        <translation type="unfinished">Кнопка2</translation>
    </message>
    <message>
        <location filename="../mainwindow.cpp" line="34"/>
        <source>TEXT_TO_TRANSLATE</source>
        <translation>Текст, который был переведён в ts-файле.</translation>
    </message>
</context>
</TS>
```