Программирование в Qt

Программирование в KDE с помощью Qt

- знакомство с комплектом инструментов Qt;
- установка Qt;
- практическое применение;
- механизм "сигнал/слот";
- виджеты Qt;
- диалоговые окна;
- создание меню и панелей инструментов с помощью KDE.

Введение в KDE и Qt

- KDE (K Desktop Environment, K-среда рабочего стола) — графическая среда рабочего стола с открытым программным кодом, основанная на библиотеке графического пользовательского интерфейса Qt. В состав KDE входит множество приложений и утилит, включая полный офисный пакет, Web-обозреватель и даже полнофункциональную IDE (интегрированная среда разработки) для программирования приложений KDE/Qt (KDevelop). Главная страница проекта KDE находится по адресу **http://www.kde.org**.

Введение в KDE и Qt

- С точки зрения программиста, KDE предлагает десятки виджетов KDE, обычно унаследованных от их аналогов Qt, но улучшенных и облегчающих использование. Виджеты KDE обеспечивают более тесную связь с рабочим столом KDE, чем сам по себе комплект инструментов Qt; у вас появляется, например, возможность управления сеансами.
- Qt — тщательно продуманный межплатформный комплект инструментов GUI, написанный на языке C++. Это детище норвежской компании Trolltech, разрабатывающей, продающей и осуществляющей техническую поддержку Qt и сопутствующего программного обеспечения для промышленного рынка.

Введение в KDE и Qt

- Qt поддерживается в ОС Linux и модификациях UNIX, Windows, Mac OS X и даже на встроенных платформах.
- Qt Open Source Edition распространяется на условиях лицензии GPL. У свободно распространяемой версии есть два основных отличия от коммерческих версий: отсутствие технической поддержки и запрет на применение программного обеспечения Qt в коммерческих приложениях. Вся необходимая документация по API есть на Web-сайте Trolltech по адресу http://www.trolltech.com.
- Русскоязычная документация: http://qtdocs.narod.ru.

Установка Qt

- Самый простой путь — найти для вашего дистрибутива двоичный пакет или пакет RPM. Дистрибутив CentOS 6.3 поставляется с пакетом qt-3.3.8-4.i386.rpm, который можно установить с помощью следующей команды.
- $ **rpm -Uvh qt-3.3.8-4.i386.rpm**
- Комплект Qt и библиотеки программирования KDE можно также установить с помощью приложения **Package Manager** (Диспетчер пакетов).
- Можно загрузить из Интернета исходный программный код и сформировать Qt самостоятельно, самый свежий программный код можно получить с FTP-сайта Trolltech по адресу **ftp://ftp.trolltech.com/qt/source/.**

Установка Qt

- Пакет исходного программного кода приходит с подробнейшими инструкциями, касающимися компиляции и установки Qt и хранящимися в файле INSTALL, упакованном программой tar.
- $ **cd /usr/local**
- $ **tar -xvzf qt-x11-free-3.3.8.tar.gz**
- $ **./configure**
- $ **make**
- Также следует добавить в файл /etc/ld.so.conf следующую строку:
- **/usr/lib/qt-3.3/lib**
- Вставить ее можно в любое место файла.

Установка Qt

- В системах Linux Fedora и Red Hat эту строку нужно сохранить в файле /etc/ld.so.conf.d/qt-i386.conf. Если вы устанавливали Qt, как показано на рис. 1, этот этап уже будет пройден.
- Если комплект Qt установлен корректно, переменная окружения QTDIR
- будет содержать каталог установки. Проверить это можно следующим образом:
- **$ echo $QTDIR**
- **/usr/lib/qt-3.3**
- Убедитесь также в том, что каталог lib добавлен в файл /etc/ld.so.conf. Затем выполните как суперпользователь следующую команду:
- **# ldconfig**

Установка Qt

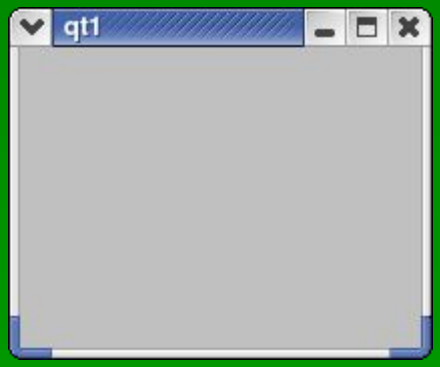
- Испытайте простейшую программу с применением Qt и убедитесь в том, что ваша установка функционирует должным образом (упражнение 1).
- Введите (или скопируйте и вставьте программный код из загруженного файла) приведенную программу qt1.cpp.
- При компиляции вам необходимо указать Qt-каталоги include и lib:
- **$ g++ -o qt1 qt1.cpp -I$QTDIR/include -L$QTDIR/lib -lqui**
- **Примечание:** На некоторых платформах в конце строки указывается библиотека -lqt. В версии Qt 3.3, тем не менее, используйте –lqui.

Установка Qt

- Есть и иной способ компиляции, когда исходники находятся в отдельном каталоге (например, hello):
- Находясь в консольном режиме, войдите в каталог *hello* и задайте команду:
- **$ qmake -project**
- для создания файла проекта, независимого от платформы (*hello.pro*), и затем задайте команду:
- **$ qmake hello.pro**
- для создания на основе файла проекта зависимого от платформы файла *makefile*.
- Выполните команду **make** для построения программы. Затем выполняйте программу, задавая команду *hello* в системе Windows или **./hello** в системе Unix и **open hello.app** в системе Mac OS X

Установка Qt

- Выполнив приложение, вы должны получить окно Qt:
- **$ ./qt1**



- **Как это работает**
- Вы должны явно включать заголовочные файлы всех используемых объектов.
- Первый объект, с которым вы встречаетесь, — QApplication. Это главный объект Qt, который вы должны сформировать, передав ему в самом начале аргументы командной строки.

Установка Qt

- У каждого приложения Qt должен быть один и только один объект типа QApplication, который вы должны создать перед тем, как делать что-то еще.
- QApplication имеет дело с внутренними встроенными операциями Qt, такими как обработка событий, размещение строк и управление внешним представлением.
- Есть два метода QApplication:
- **setMainWidget**, который создает главный виджет вашего приложения,
- и **exec**, который запускает выполнение цикла отслеживания событий.

Установка Qt

- Метод exec не возвращает управление до тех пор, пока либо не будет вызван метод QApplication::quit(), либо не будет закрыт главный виджет.
- QMainWindow — базовый виджет окна в Qt, который поддерживает меню, панель инструментов и строку состояния. Он будет играть важную роль в этой главе, по мере того, как вы научитесь расширять его возможности и вставлять в него виджеты, формирующие интерфейс.

Сигналы и слоты

- Сигналы и их обработка — главные механизмы, используемые приложениями GUI для реагирования на ввод пользователя, и ключевые функции библиотек GUI. Механизм обработки сигналов комплекта Qt состоит из сигналов и слотов или приемников, называемых сигналами и функциями обратного вызова в комплекте инструментов GTK+ или событиями и обработчиками событий в языке программирования Java.
- **Примечание:** Имейте в виду, что сигналы Qt отличаются от сигналов UNIX.

Сигналы и слоты

- Вот как устроено программирование, управляемое событиями: графический интерфейс пользователя состоит из меню, панелей инструментов, кнопок, полей ввода и множества других элементов GUI, называемых виджетами. Когда пользователь взаимодействует с виджетом, например, активизирует пункт меню или вводит какой-то текст в поле ввода, виджет порождает именованный сигнал, такой как **clicked**, **text_changed** или **key_pressed**. Как правило, требуется сделать что-то в ответ на действие пользователя, например, сохранить документ или выйти из приложения, и вы выполняете это, связав сигнал с функцией обратного вызова или слотом на языке Qt.

Сигналы и слоты

- Применение сигналов и слотов довольно специфично — Qt определяет два новых соответствующим образом описанных псевдоключевых слова, **signals** и **slots** для обозначения в вашем программном коде классов сигналов и слотов. Это замечательно с точки зрения читаемости и сопровождения программного кода, но вы вынуждены пропускать свой код через отдельный этап препроцессорной обработки для поиска и замены этих псевдоключевых слов дополнительным кодом на языке C++.
- **Примечание:** Таким образом, программный код с использованием Qt — не настоящий программный код на C++. Порой это становится проблемой для некоторых разработчиков.

Сигналы и слоты

- На способы применения сигналов и слотов в Qt есть несколько ограничений, но они не слишком существенные:
  - сигналы и слоты должны быть функциями-методами класса-потомка QObject;
  - при использовании множественного наследования QObject должен быть первым в списке класса;
  - оператор Q_OBJECT должен появляться первым в объявлении класса;
  - сигналы нельзя применять в шаблонах;
  - указатели на функцию не могут использоваться как аргументы в сигналах и слотах;
  - сигналы и слоты не могут переопределяться или обновляться до статуса public (общедоступный).

Сигналы и слоты

- Поскольку вы должны писать ваши сигналы и слоты как потомков объекта QObject, логично создавать ваш интерфейс, расширяя и настраивая виджет, начиная с QWidget, базового виджета Qt, потомка виджета QObject. В комплекте Qt вы почти всегда будете создавать интерфейсы, расширяя такие виджеты, как QMainWindow.
- Типичное определение класса в файле MyWindow.h для вашего GUI будет напоминать приведенное далее:

Сигналы и слоты

```
class MyWindow : public QMainWindow {
 Q_OBJECT
public:
 MyWindow();
 virtual ~MyWindow();
signals:
 void aSignal();
private slots:
 void doSomething();
}
```

Сигналы и слоты

- Ваш класс — наследник объекта QMainWindow, который определяет функциональные возможности главного окна в приложении. Аналогичным образом при создании диалогового окна вы определите подкласс QDialog. Первым указан оператор Q_OBJECT, действующий как метка для препроцессора, за которым следуют обычные объявления конструктора и деструктора. Далее даны определения сигнала и слота.
- У вас есть один сигнал и один слот, оба без параметров. Для порождения сигнала aSignal() вам нужно всего лишь в любом месте программы вызвать функцию **emit**:
- **emit aSignal();**

Сигналы и слоты

- Это означает, что все остальное обрабатывается Qt. Вам даже не потребуется реализация aSignal().
- Для применения слотов их нужно связать с сигналом. Делается это соответствующим образом с помощью названного статического метода **connect** класса QObject:
- **bool QObject::connect(const QObject * sender, const char* signal,**
- **const QObject * receiver, const char * member);**
- Просто передайте объект, владеющий сигналом (отправитель), функцию сигнала, объект, владеющий слотом (приемником), и в завершение укажите имя слота.

Сигналы и слоты

- В примере MyWindow, если бы вы захотели связать сигнал clicked виджета QPushButton с вашим слотом doSomething, вы бы написали:
- **connect(button, SIGNAL(clicked()), this, SLOT(doSomething()));**
- Учтите, что необходимо применять макросы SIGNAL и SLOT для выделения функций сигналов и слотов. Вы можете связать ряд слотов с заданным сигналом и также связать слот с любым количеством сигналов с помощью множественных вызовов функции **connect**. Если она завершается аварийно, то возвращает FALSE.

Сигналы и слоты

- Остается реализовать ваш слот в виде обычной функции-метода:
- **void MyWindow::doSomething() {**
- **// Код слота**
- **}**
- **Упражнение 2. Сигналы и слоты**
- Теперь, зная основы использования сигналов и слотов, применим их в примере. Усовершенствуйте QMainWindow, вставьте в него кнопку и свяжите сигнал кнопки clicked со слотом.
- 1. Возьмите следующее объявление класса из файла ButtonWindow.h

Сигналы и слоты

- Далее следует реализация класса в файле ButtonWindow.cpp.
- В конструкторе вы задаете заголовок окна, создаете кнопку и связываете сигнал нажатия кнопки с вашим слотом.
- setCaption — метод объектов типа QMainWindow, который, что неудивительно, задает заголовок окна.
- Qt автоматически удаляет виджеты, поэтому ваш деструктор пуст.
- Затем реализация слота:

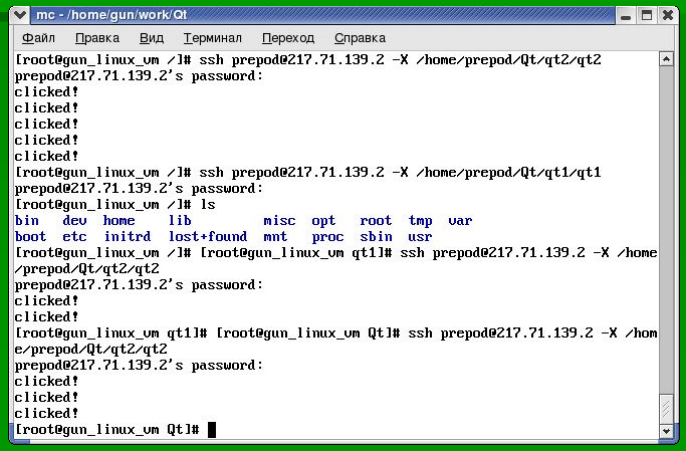
```
void ButtonWindow::Clicked(void) {
  std::cout << "clicked!\n";
}
```

Сигналы и слоты

- И наконец, в функции main вы просто создаете экземпляр типа ButtonWindow, делаете его главным окном вашего приложения и отображаете окно на экране.
- Прежде чем вы сможете откомпилировать данный пример, необходимо запустить препроцессор для заголовочного файла. Программа этого препроцессора называется Meta Object Compiler (moc, компилятор метаобъекта) и должна быть включена в пакет комплекта Qt. Выполните **moc** для файла ButtonWindow.h, сохранив результат в файле ButtonWindow.moc:
- **$ moc -o ButtonWindow.moc ButtonWindow.h**

Сигналы и слоты

- Теперь можно компилировать как обычно, скомпоновав с результатом команды moc.
- **$ g++ -o button ButtonWindow.cpp -I$QTDIR/include -L$QTDIR/lib -lqui**
- Выполнив программу, вы получите пример, показанный на рис.
- При использовании **qmake** в cpp-файле надо подключать не moc-файл, а h-файл.



Сигналы и слоты

- **Как это работает**
- В этом примере мы ввели новый виджет и некоторые новые функции:
- QPushButton — виджет простой кнопки, хранящий метку и растровую графику и способный активизироваться при щелчке пользователя кнопкой мыши или при нажатии клавиш.
- Конструктор объекта QPushButton очень прост.
- QPushButton::QPushButton(const QString &text, QWidget *parent, const char* name=0);
- Первый аргумент — текст метки кнопки, далее родительский виджет и последний аргумент — имя кнопки, обычно применяемое Qt для внутренних операций.

Сигналы и слоты

- Параметр родительского виджета, общий для всех объектов, — QWidget, он управляет отображением и уничтожением и разными другими свойствами. Передача NULL в качестве родительского объекта означает виджет верхнего уровня, при этом создается содержащее его пустое окно. В примере вы передаете текущий объект ButtonWindow с помощью ключевого слова this, что приводит к вставке кнопки в основную область окна ButtonWindow.
- Аргумент **name** задает имя виджета для внутреннего использования Qt. Если Qt обнаружит ошибку, имя виджета будет выведено в сообщении об ошибке.

Сигналы и слоты

- Объект QPushButton очень примитивно вставляется в окно ButtonWindow, с помощью параметра parent конструктора QPushButton, без указания положения кнопки, ее размера, рамки или чего-либо еще. Если вы хотите управлять внешним видом кнопки, что очень важно для создания привлекательного интерфейса, следует применять виджеты компоновки комплекта Qt.
- В Qt есть целый ряд способов размещения и компоновки виджетов. Возможно использование абсолютных координат с помощью вызова setGeometry, но они редко применяются, поскольку виджеты не масштабируются и не меняют размеры при изменении величины окна.

Сигналы и слоты

- Предпочтительный метод компоновки виджетов — применение классов QLayout или виджетов-контейнеров, которые изменяют свои размеры соответствующим образом после задания им подсказок, касающихся отступов и расстояний между виджетами.
- Ключевое различие между классами QLayout и упаковочными контейнерами заключается в том, что объекты класса QLayout *не* являются виджетами.
- Классы компоновки — потомки объектов, типа QObject, а не QWidget, поэтому их применение ограничено. Например, вы не можете создать объект QVBoxLayout — основной виджет объекта QMainWindow.

Сигналы и слоты

- Виджеты упаковочных контейнеров (такие, как QHBox и QVBox) напротив — потомки объекта типа QWidget, следовательно, вы можете применять их как обычные виджеты. На самом деле виджеты QBox существуют только для удобства и по существу служат оболочкой классов QLayout в типе QWidget. Объекты QLayout обладают возможностью автоматического изменения размеров, в то время как размеры виджетов нужно изменять вручную с помощью вызова метода QWidget::resizeEvent().
- Подклассы QLayout: QVBoxLayout и QHBoxLayout, — самый распространенный способ создания интерфейса.

Сигналы и слоты

- QVBoxLayout и QHBoxLayout — невидимые объекты-контейнеры, хранящие другие виджеты и схемы размещения с вертикальной и горизонтальной ориентациями соответственно. Вы сможете создавать сколь угодно сложные компоновки виджетов, поскольку допускается использование вложенных компоновок, например, за счет вставки как элемента горизонтальной схемы размещения внутрь вертикального упаковочного контейнера.
- Есть три конструктора QVBoxLayout, заслуживающих внимания (у объектов QHBoxLayout идентичный API).

Сигналы и слоты

- **QVBoxLayout::QVBoxLayout(QWidget *parent, int margin, int spacing, const char *name)**
- **QVBoxLayout::QVBoxLayout(QLayout *parentLayout, int spacing, const char * name)**
- **QVBoxLayout::QVBoxLayout(int spacing, const char *name)**
- Родителем объекта QLayout может быть либо виджет, либо другой объект типа QLayout. Если не задавать родительский объект, вы сможете только вставить позже данную схему размещения в другой объект QLayout с помощью метода addLayout.
- Параметры margin и spacing задают пустое пространство в пикселах вокруг схемы размещения QLayout и между отдельными виджетами в ней.

Сигналы и слоты

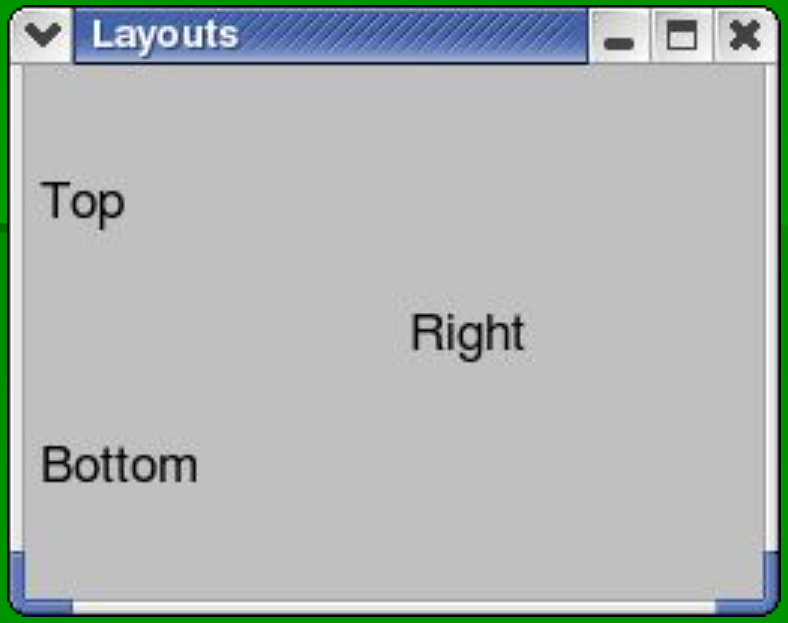
- После создания вашей схемы размещения QLayout вы можете вставлять дочерние виджеты или схемы с помощью следующей пары методов:
- **QBoxLayout::addWidget(QWidget *widget, int stretch = 0, int alignment = 0);**
- **QBoxLayout::addLayout(QLayout *layout, int stretch = 0);**
- Выполните упражнение 3.
- В этом примере вы увидите в действии классы QBoxLayout при размещении виджетов QLabel в окне QMainWindow.
- Сначала скопируйте заголовочный файл LayoutWindow.h.

Сигналы и слоты

- Теперь скопируйте реализацию из файла LayoutWindow.cpp.
- В нем необходимо создать фиктивный QWidget для хранения объекта QHBoxLayout, поскольку его нельзя напрямую вставить в объект QMainWindow:
- `QWidget *widget = new QWidget(this);`
- `QHBoxLayout *horizontal = new QHBoxLayout(widget, 5, 10, "horizontal");`
- Как и прежде, перед компиляцией нужно выполнить **moc** для заголовочного файла:
- **$ moc LayoutWindow.h -o LayoutWindow.moc**
- **$ g++ -o layout LayoutWindow.cpp -I$QTDIR/include -L$QTDIR/lib -lqui**

Сигналы и слоты

- Выполнив эту программу, вы получите схему размещения ваших меток QLabel (см. рис.). Попробуйте изменить величину окна и посмотрите, как расширяются и сжимаются метки, заполняя все доступное пространство.



Сигналы и слоты

- **Как это работает**
- Программа LayoutWindow.cpp создает два виджета упаковочных контейнеров, горизонтальный и вертикальный контейнер для размещения виджетов. Вертикальный контейнер получает две метки, описанные, соответственно, как Top и Bottom. Горизонтальный контейнер также содержит два виджета, метку, обозначенную Right, и вертикальный контейнер. Вы можете помещать компоновочные виджеты внутрь других компоновочных виджетов, как показано в данном примере.
- Мы рассмотрели основы применения Qt — сигналы и слоты, команду moc и средства компоновки. Теперь пора более внимательно изучить виджеты.

Виджеты Qt

- Для каждого случая в Qt есть виджеты, и их подробное рассмотрение займет всю оставшуюся часть жизни. В этом разделе мы познакомимся с виджетами Qt общего применения, включая виджеты для ввода данных, кнопки, обычные и раскрывающиеся списки.
- **QLineEdit** — виджет для ввода однострочного В виджете QLineEdit можно ограничить длину вводимого текста с помощью маски ввода, предлагающей заполнить шаблон, или для дополнительного контроля можно применить функцию проверки допустимости, что пользователь вводит корректные дату, номер телефона или подобные величины.

Виджеты Qt

- У виджета QLineEdit есть функции редактирования, позволяющие выбирать части текста, вырезать и вставлять, отменять и повторять изменения, как командами пользователя, так и средствами API.
- Далее перечислены конструкторы и наиболее полезные методы.
- **#include <qlineedit.h>**
- **QLineEdit::QLineEdit(QWidget *parent, const char* name = 0);**
- **QLineEdit::QLineEdit(const QString& contents, QWidget *parent,**
- **const char *name = 0);**

Виджеты Qt

- QLineEdit::QLineEdit(const QString& contents, const QString& inputMask, QWidget *parent, const char* name = 0);
- void setInputMask(const QString& inputMask);
- void insert(const QString& newText);
- bool isModified(void);
- void setMaxLength(int length);
- void setReadOnly(bool read);
- void setText(const QString &text);
- QString text(void);
- void setEchoMode(EchoMode mode);

Виджеты Qt

- В конструкторах вы задаете родительский виджет и имя виджета с помощью параметров parent и name.
- Интересно свойство EchoMode, определяющее способ отображения текста в виджете. Оно может принимать одно из трех значений:
- QLineEdit::Normal — отображать вводимые символы (по умолчанию);
- QLineEdit::Password — отображать звездочки на месте символов;
- QLineEdit::NoEcho — ничего не отображать. Задается режим отображения с помощью метода setEchoMode:lineEdit->setEchoMode(QLineEdit::Password);

Виджеты Qt

- Усовершенствование, внесенное в версию Qt 3.2, — свойство inputMask, ограничивающее ввод в соответствии с правилом маски.
- inputMask — это строка, сформированная из символов, каждый из которых соответствует правилу, принимающему диапазон определенных символов. Если вы знакомы с регулярными выражениями, inputMask использует во многом тот же самый принцип.
- Есть два сорта символов, формирующих inputMask: первые указывают на необходимость присутствия определенного символа, вторые при наличии символа добиваются его соответствия заданному правилу. В табл.1 приведены примеры таких символов и их значения.

Виджеты Qt

Обязательный символ	Символы, разрешены, но не обязательны	Значение
A	a	A–Z, a–z
N	n	A–Z, a–z, 0–9
X	x	Любой символ
9	0	Цифры 0–9
D	d	Цифры 1–9

Виджеты Qt

- Наша inputMask — это строка, сформированная комбинацией этих символов и необязательно завершающаяся точкой с запятой. Существуют дополнительные специальные символы, у которых также есть значения (табл. 2).
- Все остальные символы в inputMask действуют как разделители в поле ввода QLineEdit.
- В табл. 3 приведены примеры масок ввода и соответствующий им текст для ввода.

Виджеты Qt

Символ	Значение
#	Разрешен, но не обязателен знак +/-
>	Преобразует все последующие символы в верхний регистр.
<	Преобразует все последующие символы в нижний регистр
!	Останавливает преобразование регистра
\	Символ управляющей последовательности

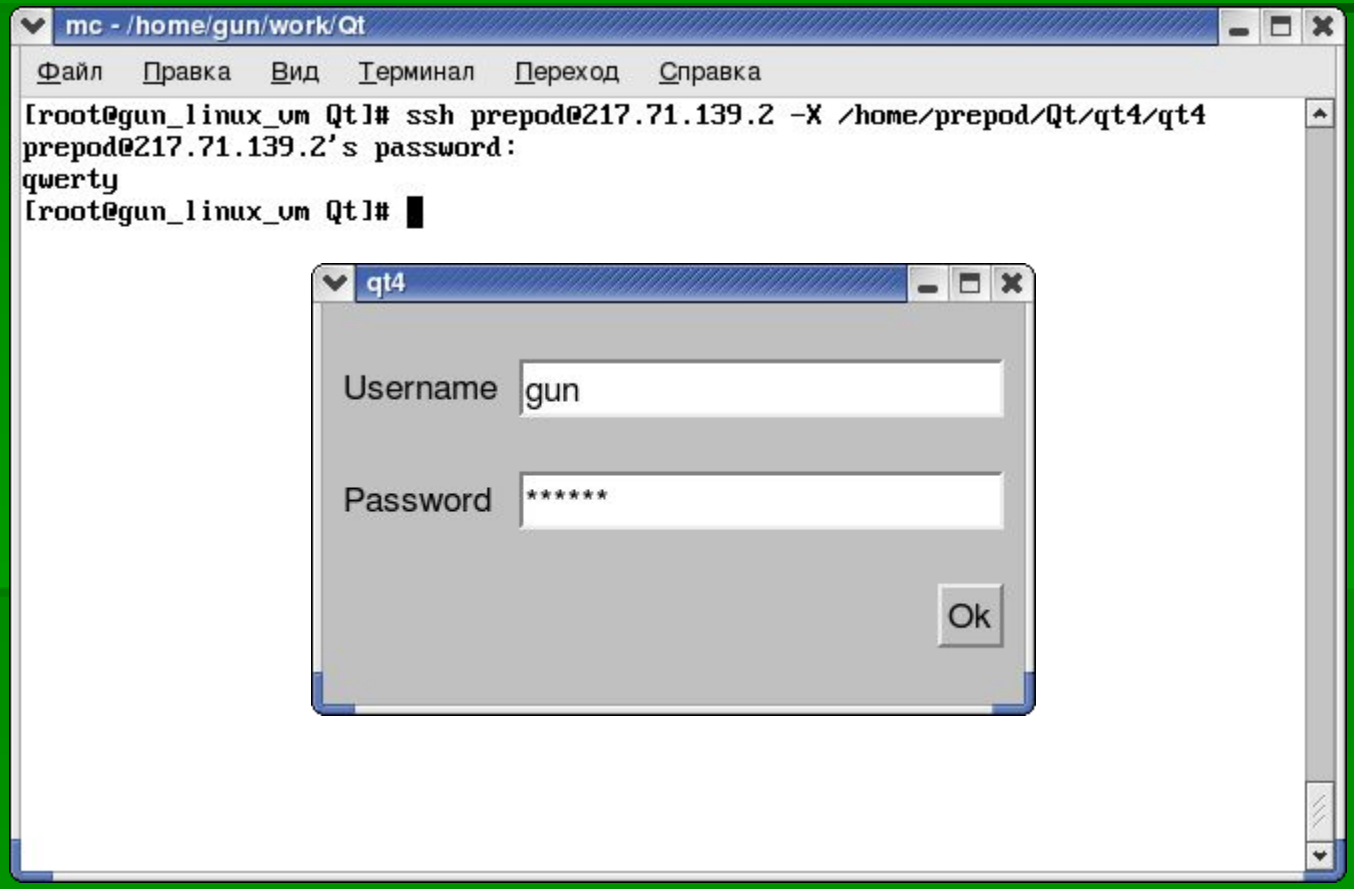
Виджеты Qt

Пример	Допустимый ввод
"AAAAAA-999D"	Athens-2004, но не Sydney-2000 или Atlanta-1996
"AAAAnn-99-99;"	March-03-12, но не May-03-12 или September-03-12
"000.000.000.000"	IP-адрес, например, 192.168.0.1

Виджеты Qt

- Выполните упражнение 4.
- Посмотрим, как действует виджет QLineEdit.
- Сначала — заголовочный файл LineEdit.h.
- LineEdit.cpp —файл реализации класса.
- Для компоновки виджетов применен QGridLayout. Задано число строк и столбцов, величины отступов и расстояния между виджетами
- Выполнив эту программу, вы должны получить результат, представленный далее:

Виджеты Qt


mc - /home/gun/work/Qt
Файл Правка Вид Терминал Переход Справка
[root@gun_linux_vm Qt]# ssh prepod@217.71.139.2 -X /home/prepod/Qt/qt4/qt4
prepod@217.71.139.2's password:
qwerty
[root@gun_linux_vm Qt]#
qt4
Username
gun
Password
******
Ok

Виджеты Qt

- **Как это работает**
- Вы создали два виджета QLineEdit, один подготовили для ввода пароля, задав EchoMode, и заставили его выводить содержимое при щелчке мышью кнопки PushButton. Обратите внимание на виджет QGridLayout, который очень полезен для размещения виджетов в табличной сетке. Когда виджет вставляется в сетку таблицы, вы передаете номер строки и столбца, нумерация начинается с 0, нулевые номера строки и столбца у верхней левой ячейки.

Виджеты Qt

- **Кнопки Qt**
- Кнопки виджетов вездесущи и мало отличаются внешним видом, способом применения и API в разных комплектах инструментов. Неудивительно, что Qt предлагает стандартные кнопки PushButton, флажки CheckBox и радиокнопки (или зависимые переключатели) RadioButton.
- ***QButton:* базовый класс кнопок**
- Все виджеты кнопок в комплекте Qt — потомки абстрактного класса QButton. У этого класса есть методы для опроса и переключения включенного/выключенного состояния кнопки и задания текста кнопки или ее графического представления.

Виджеты Qt

- Вам никогда не придется обрабатывать виджет типа QButton (не путайте с виджетом QPushButton!), поэтому нет смысла приводить конструкторы. Далее перечислено несколько полезных функций-методов этого класса:
- **#include <qbutton.h>**
- **virtual void QButton::setText(const QString&);**
- **virtual void QButton::setPixmap(const QPixmap&);**
- **bool QButton::isToggleButton() const;**
- **virtual void QButton::setDown(bool);**
- **bool QButton::isDown() const;**
- **bool QButton::isOn() const;**
- **enum QButton::ToggleState { Off, NoChange, On }**
- **ToggleState QButton::state() const;**

Виджеты Qt

- Функции isDown и isOn возвращают TRUE, если кнопка была нажата или активизирована.
- Сделать недоступным любой виджет, включая QButton, можно с помощью вызова метода QWidget::setEnable(FALSE).
- У QButton есть три подкласса, заслуживающие внимания:
- QPushButton — виджет простой кнопки, выполняющий некоторое действие при щелчке кнопкой мыши;
- QCheckBox — виджет кнопки, способный изменять состояние с включенного на выключенное для обозначения некоторого выбора;

Виджеты Qt

- QRadioButton — виджет кнопки, обычно применяемый в группе таких же кнопок, только одна из которых может быть активна в любой момент времени.
- **QPushButton** — стандартная кнопка общего вида, содержащая текст, такой как "OK" или "Cancel" и/или пиксельную пиктограмму. Как все кнопки класса QButton, она порождает при активизации сигнал **clicked** и обычно используется для связи со слотом и выполнения некоторого действия.
- Кнопку QPushButton можно превратить из кнопки, не помнящей своего состояния, в кнопку-выключатель (т.е. способную быть включенной и выключенной), вызвав метод setToggleButton.

Виджеты Qt

- Далее для полноты описания приведены конструкторы и полезные методы.
- **#include <qpushbutton.h>**
- **QPushButton(QWidget *parent, const char *name = 0);**
- **QPushButton(const QString& text, QWidget *parent, const char *name = 0);**
- **QPushButton(const QIconSet& icon, const QString& text,**
- **QWidget *parent, const char * name = 0);**
- **void QPushButton::setToggleButton(bool);**

Виджеты Qt

- **QCheckBox** — это кнопка, у которой есть состояние, ее можно установить и сбросить. Внешний вид QCheckBox зависит от стиля отображения окон текущей системы (Motif, Windows и т.д.), но обычно она отображается как флажок с сопроводительным текстом справа.
- Вы можете также перевести кнопку QCheckBox в третье промежуточное состояние, которое означает "без изменения". Оно бывает полезно в редких случаях, когда вы не можете прочесть состояние выбора, который предоставляет кнопка QCheckBox (и, следовательно, самостоятельно установить или сбросить флажок), но хотите дать пользователю возможность оставить выбор неизменным наряду с установкой и сбросом.

Виджеты Qt

- Конструкторы и полезные методы:
- **#include <qcheckbox.h>**
- **QCheckBox(QWidget *parent, const char *name = 0);**
- **QCheckBox(const QString& text, QWidget *parent, const char *name = 0);**
- **bool QCheckBox::isChecked();**
- **void QCheckBox::setTristate(bool y = TRUE);**
- **bool QCheckBox::isTristate();**

Виджеты Qt

- ***QRadioButton***
- Радиокнопки — кнопки-переключатели, применяемые для отображения исключающего выбора, когда можно выбрать только один вариант из группы представленных (вспомните снова старые автомобильные радиоприемники, в которых можно было нажать только одну кнопку блока). Сами по себе кнопки QRadioButton не многим отличаются от кнопок QCheckBox, поскольку группировка и исключительный выбор обрабатываются классом QButtonGroup, главное же их отличие заключается в том, что они отображаются как круглые кнопки, а не как флажки.

Виджеты Qt

- QButtonGroup — виджет, облегчающий обработку групп кнопок.
- **#include <qbuttongroup.h>**
- **QButtonGroup(QWidget *parent = 0, const char* name = 0);**
- **QButtonGroup(const QString& title, QWidget* parent = 0,**
- **const char * name = 0);**
- **int insert (QButton *button, int id = -1);**
- **void remove(QButton *button);**
- **int id(QButton *button) const;**
- **int count() const;**
- **int selectedId() const;**

Виджеты Qt

- Применять виджет QButtonGroup проще простого: он даже предлагает необязательную рамку вокруг кнопок, если используется конструктор title.
- Добавить кнопку в QButtonGroup можно с помощью метода **insert** или заданием QButtonGroup в качестве родительского виджета кнопки. Для уникального обозначения каждой кнопки в группе можно задать **id** в методе **insert**. Это особенно полезно при определении выбранной кнопки, т.к. функция **selectedId** возвращает **id** выбранной кнопки.
- Все кнопки QRadioButton, добавляемые в группу, автоматически становятся кнопками с исключающим выбором.

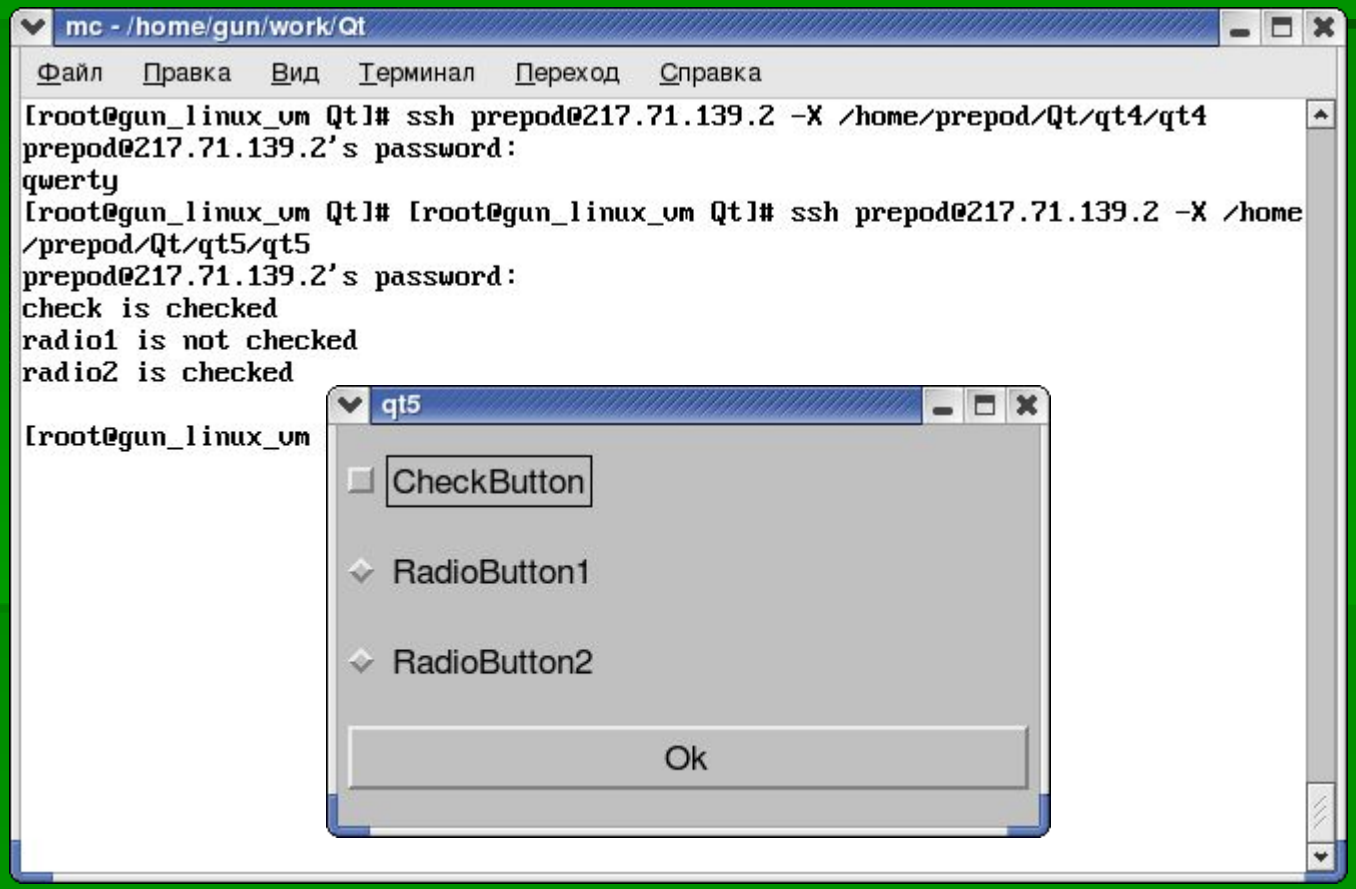
Виджеты Qt

- Далее приведены прототипы конструкторов QRadioButton и одного уникального метода:
- **#include <qradiobutton.h>**
- **QRadioButton(QWidget* parent, const char* name = 0);**
- **QRadioButton(const QString& text, QWidget *parent, const char *name = 0);**
- **bool QRadioButton::isChecked();**

Виджеты Qt

- Выполните упражнение 5.
- Теперь применим полученные знания в примере с кнопками Qt. Приведенная далее программа создает кнопки разных типов (радиокнопки, флажки и простые кнопки), чтобы показать, как использовать эти виджеты в ваших приложениях.
- Скопируйте файл Buttons.h.
- Вы запросите состояние ваших кнопок позже, в функции слота, поэтому объявите указатели кнопок и вспомогательную функцию PrintActive с атрибутом **private** в объявлении класса.
- Далее следует файл Buttons.Далее следует файл Buttons.cpp.
- Пример работы представлен далее.

Виджеты Qt


mc - /home/gun/work/Qt
Файл Правка Вид Терминал Переход Справка
[root@gun_linux_vm Qt]# ssh prepod@217.71.139.2 -X /home/prepod/Qt/qt4/qt4
prepod@217.71.139.2's password:
qwerty
[root@gun_linux_vm Qt]# [root@gun_linux_vm Qt]# ssh prepod@217.71.139.2 -X /home
/prepod/Qt/qt5/qt5
prepod@217.71.139.2's password:
check is checked
radio1 is not checked
radio2 is checked
[root@gun_linux_vm
qt5
CheckButton
RadioButton1
RadioButton2
Ok

Виджеты Qt

- **Как это работает**
- Функция PrintActive демонстрирует, как получить состояние кнопки (включена или выключена). Обратите внимание на то, как она действует в случае запоминающих состояние кнопок разных типов, таких как флажки и переключатели (радиокнопки). В основном отличаются только вызовы для создания виджета кнопки. Радиокнопки, наиболее сложные (т.к. только одна в группе может быть включена), при создании требуют больше всего работы. В случае радиокнопок вы должны создать QButtonGroup для того, чтобы гарантировать активность только одной радиокнопки в группе в любой момент времени.

Виджеты Qt

- ***QComboBox***
- Переключатели (радиокнопки) — отличный способ, позволяющий пользователю выбрать из небольшого числа вариантов. Если вариантов больше шести, ситуация начинает выходить из-под контроля и становится еще более напряженной, когда количество вариантов растет, что приводит к ощутимому увеличению размера окна. В этом случае прекрасным решением может быть использование поля ввода с раскрывающимся меню, также называемое раскрывающимся списком (combo box). Варианты выводятся, когда вы щелкаете кнопкой мыши и открываете меню и количество вариантов при этом ограничено только удобством поиска в списке.

Виджеты Qt

- В виджете QComboBox сочетаются функциональные возможности виджетов QLineEdit и QPushButton и раскрывающихся меню, позволяя выбрать один вариант из неограниченного набора вариантов.
- QComboBox может быть открытым, как для чтения и записи, так и только для чтения. Если он позволяет читать и записывать, пользователь может ввести новый вариант в дополнение к предлагаемым; в противном случае пользователь ограничен выбором варианта из раскрывающегося списка.

Виджеты Qt

- При создании виджета QComboBox можно указать, открыт ли он для чтения и записи или только для чтения, задавая логическое значение в конструкторе:
- QComboBox *combo = new QComboBox(TRUE, parent, "widgetname");
- Передача значения TRUE переводит QComboBox в режим "чтение/запись". Остальные параметры — обычный указатель на родительский виджет и имя создаваемого виджета.
- Вы можете добавлять варианты по одному или набором, как тип QString или в стандартном формате char*.

Виджеты Qt

- Для вставки одного варианта вызовите функцию **insertItem**:
- combo->insertItem(QString("An Item"), 1);
- Приведенная функция принимает объект типа QString и номер позиции в списке. В данном случае 1 вставляет вариант в список первым.
- Для добавления в конец списка задайте любое отрицательное целое число.
- Гораздо чаще вы будете вставлять несколько элементов списка одновременно, для этого можно применить класс QStrList или, как показано далее, массив char*:
- char* weather[] = {"Thunder", "Lightning", "Rain", 0};
- combo->insertStrList(weather, 3);

Виджеты Qt

- И снова вы можете задать номер позиции вставляемых в список элементов.
- Если в виджете QComboBox задан режим "чтение/запись", вводимые пользователем варианты могут автоматически вставляться в список. Это очень полезное, экономящее время свойство, избавляющее пользователя от повторного набора варианта, если он хочет уже введенный вариант использовать несколько раз.
- Метод **InsertionPolicy** управляет позицией вводимого в список элемента. Вы можете выбрать одно из значений, приведенных в табл. 4.

Виджеты Qt

Значение	Действие
QComboBox::AtTop	Вставляет элемент первым
QComboBox::AtBottom	Вставляет элемент последним
QComboBox::AtCurrent	Заменяет предварительно выбранный вариант в списке
QComboBox::BeforeCurrent	Вставляет элемент перед предварительно выбранным вариантом из списка
QComboBox::AfterCurrent	Вставляет элемент после предварительно выбранного варианта из списка
QComboBox::NoInsertion	Новый элемент не вставляется в список вариантов

Виджеты Qt

- Для задания политики вызовите метод InsertionPolicy виджета QComboBox:
- combo->setInsertionPolicy(QComboBox::AtTop);
- Конструкторы и методы выбора варианта виджета QComboBox:
- **#include <qcombobox.h>**
- **QComboBox(QWidget *parent = 0, const char *name = 0);**
- **QComboBox(bool readwrite, QWidget *parent = 0, const char *name = 0);**
- **int count();**
- **void insertStringList(const QStringList& list, int index = -1);**

Виджеты Qt

- void insertStrList(const QStrList& list, int index = -1);
- void insertStrList(const QStrList *list, int index = -1);
- void insertStrList (const char **strings, int numStrings = -1, int index = -1);
- void insertItem(const QString &t, int index = -1);
- void removeItem(int index);
- virtual void setCurrentItem(int index);
- QString currentText();
- virtual void setCurrentText(const QString &);
- void setEditable(bool);

Виджеты Qt

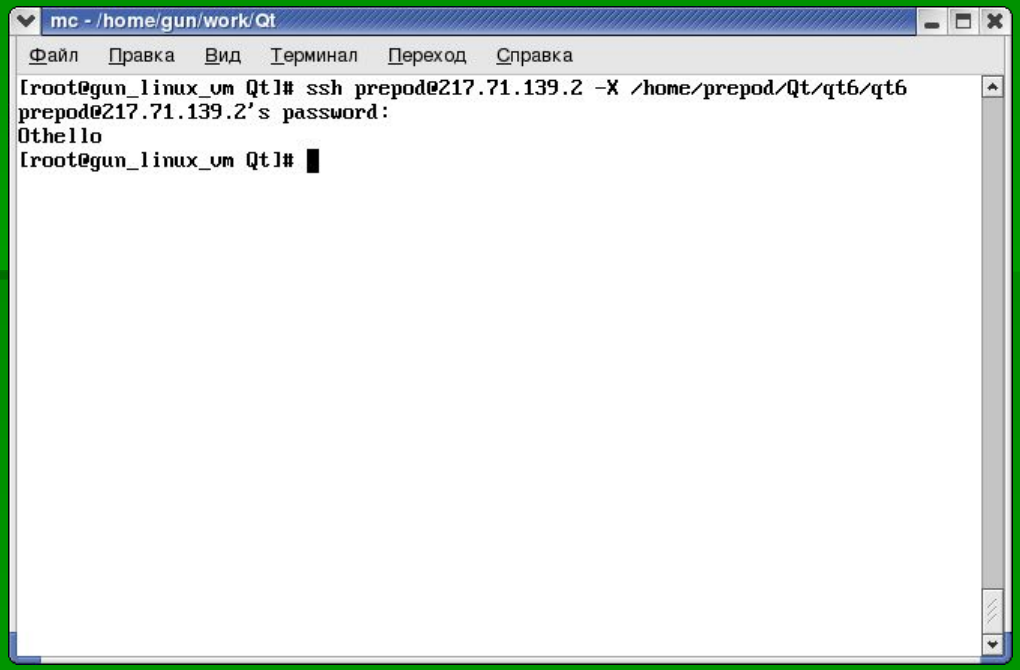
- Функция **count** возвращает количество вариантов в списке.
- **QStringList** и **QStrList** — классы коллекций, которые можно применять для вставки вариантов. Удалить варианты можно с помощью метода removeItem, извлечь и задать текущий вариант можно, с помощью методов **currentText** и **setCurrentText**, а перейти в редактируемый режим — с помощью метода **setEditable**.
- QComboBox порождает сигнал **textChanged**(QString&) при каждом новом выборе варианта, передавая вновь выбранный элемент как аргумент.
- Выполните упражнение 6.

Виджеты Qt

- В этом примере вы сделаете попытку применить виджет QComboBox и посмотрите, как ведут себя сигналы и слоты с параметрами. Вы создадите класс ComboBox, потомка QMainWindow. В нем будут два виджета QComboBox: один для чтения/записи, другой только для чтения. Вы установите связь с сигналом textChanged для того, чтобы получать текущее значение при каждом его изменении.
- Скопируйте код из файла ComboBox.h.
- Интерфейс состоит из двух виджетов QComboBox: один редактируемый, а другой предназначен только для чтения. Вы заполните оба списка одними и теми же вариантами.

Виджеты Qt

- Далее идет функция слота. Обратите внимание на параметр s типа QString, передаваемый сигналом.
- Вы сможете видеть вновь выбранные из редактируемого QComboBox варианты в командной строке.



Виджеты Qt

- **Как это работает**
- Создаются виджеты раскрывающегося списка во многом так же, как и другие виджеты. Главная новая деталь — вызов функции **insertStrList** для сохранения списка вариантов в виджете.
- Как и в других содержащих текст виджетах, можно задать функцию, которая будет вызываться каждый раз при изменении значения или в общем случае текста раскрывающегося списка.

Виджеты Qt

- ***QListView***
- Списки и деревья в комплекте Qt формируются виджетом QListView. Этот виджет представляет как простые списки, так и иерархические данные, разделенные на строки и столбцы. Он очень подходит для вывода структур каталогов или чего-то подобного, потому что дочерние элементы можно раскрыть и свернуть, щелкнув кнопкой мыши знак "плюс" или "минус", так же как в файловом обозревателе.
- QListView обрабатывает и данные, и их представление, что сделано для облегчения использования, если не для исключительной гибкости.

Виджеты Qt

- В виджете QListView можно выбрать строки или отдельные ячейки и затем вырезать и вставить данные, отсортировать их по столбцу и вы получите виджеты QCheckBox, отображенные в ячейках. В этот виджет встроено множество функциональных возможностей — как программисту вам достаточно лишь вставить данные и задать некоторые правила форматирования.
- Создается виджет QListView обычным образом, заданием родительского виджета и собственного имени виджета:
- QListView *view = new QListView(parent, "name");

Виджеты Qt

- Для задания заголовков столбцов используйте соответствующим образом названный метод **addColumn**:
- view->addColumn("Left Column", width1); // фиксированной ширины
- view->addColumn("Right Column"); // с автоматически задаваемым размером
- Ширина столбца задается в пикселах или, если пропущена, приравнивается к величине самого длинного элемента в столбце. В дальнейшем при вставке и удалении элементов ширина столбца автоматически меняется.

Виджеты Qt

- Данные вставляются в QListView с помощью объекта **QListViewItem**, представляющего строку данных. Вы должны лишь передать в конструктор объект QListView и элементы строки, и она добавится в конец представления:
- QListViewItem *toplevel = new QListViewItem(view, "Left Data", "Right Data");
- Первый параметр — либо объект QListView, как в данном случае, либо еще один объект типа QListViewItem. Если передается QListViewItem, строка создается как дочерняя по отношению к этому объекту QListViewItem. Структура дерева формируется передачей объекта QListView для узлов верхнего уровня и затем последующих объектов типа QListViewItem для дочерних узлов.

Виджеты Qt

- Остальные параметры — данные каждого столбца, по умолчанию равные NULL, если не заданы.
- Добавление дочернего узла — это просто вариант передачи в функцию указателя верхнего уровня. Если вы не добавляете последующие дочерние узлы в объект QListViewItem, нет необходимости сохранять возвращаемый указатель:
- new QListViewItem(toplevel, "Left Data", "Right Data");
- // Дочерний по отношению к верхнему уровню
- В API QListViewItem можно найти методы обхода узлов дерева на случай корректировки конкретных строк:

Виджеты Qt

- #include <qlistview.h>
- virtual void insertItem(QListviewitem* newChild);
- virtual void setText(int column, const QString& text);
- virtual QString text(int column) const;
- QListViewItem* firstChild() const;
- QListViewItem* nextSibling() const;
- QListViewItem* parent() const;
- QListViewItem* itemAbove();
- QListViewItem *itemBelow();

Виджеты Qt

- Получить первую строку в дереве можно, вызвав метод **firstChild** для самого объекта QListView. Затем можно многократно вызывать **firstChild** и **nextSibling** для возврата фрагментов или целого дерева.
- Приведенный далее фрагмент программного кода выводит первый столбец всех узлов верхнего уровня:
- QListViewItem *child = view->firstChild();
- while(child) {
- cout << myChild->text(1) << "\n";
- myChild = myChild->nextSibling();
- }

Виджеты Qt

- Все подробности, касающиеся QListView, QListViewItem и QCheckListView, см. в документации API комплекта Qt.
- Выполните упражнение 7.
- В этом упражнении вы соберете все вместе и напишете короткий пример использования виджета QListView.
- Скопируем заголовочный файл ListView.Скопируем заголовочный файл ListView.hСкопируем заголовочный файл ListView.h и рассмотрим реализацию класса, файл ListView.cpp.
- Внешний вид представлен далее.

Виджеты Qt


qt7
Artist
Title
Catalogue
Avril Lavigne
Let Go
AVCD01
Complicated
Sk8er Boi

Виджеты Qt

- **Как это работает**
- Виджет QListView кажется сложным, потому что он действует и как список элементов, и как дерево элементов. В вашем программном коде необходимо создать экземпляры QListViewItem для каждого элемента, включаемого вами в список. В этом примере показаны экземпляры QListViewItem со всего одним уровнем глубины.
- Обратите внимание на то, как дочерние строки почтительно отступают от своих "родителей". Знаки "плюс" и "минус", указывающие на наличие скрытых или сворачивающихся строк, не представлены по умолчанию; в этом примере они задаются с помощью setRootIsDecorated.

Диалоговые окна

- До сих пор вы создавали подклассы QMainWindow для построения своих интерфейсов. Объекты QMainWindow предназначены для создания главного окна в приложении, но для кратковременных диалоговых окон следует рассмотреть виджет QDialog.
- Диалоговые окна хороши для ввода пользователем определенной информации, предназначенной для конкретной задачи, или передачи пользователю коротких сообщений, таких как предупреждение или сообщение об ошибке. Для таких задач лучше применять подкласс QDialog, поскольку вы получаете удобные методы формирования диалогового окна и специализированные сигналы и слоты для обработки ответов пользователя.

Диалоговые окна

- Наряду с обычными модальными и немодальными (или безмодальными на языке Qt) диалоговыми окнами комплект Qt также предлагает полумодальное диалоговое окно. В следующем перечне приведены отличия модальных и немодальных диалоговых окон, в него также включены полумодальные окна.
- *Модальное диалоговое окно* блокирует ввод во все другие окна, чтобы заставить пользователя дать ответ в диалоговом окне. Модальные диалоговые окна полезны для захвата немедленного ответа пользователя и отображения важных сообщений об ошибках.

Диалоговые окна

- *Немодальное диалоговое окно* — неблокирующее окно, которое действует обычно наряду с другими окнами приложения. Немодальные диалоговые окна удобны для окон поиска или ввода, в которых вы сможете, например, копировать и вставлять значения в главное окно и из него.
- *Полумодальное диалоговое окно* — это модальное окно, не имеющее своего цикла событий. Это позволяет возвращать управление приложению, но сохранять блокировку ввода для других окон. Полумодальные окна бывают полезны в редких случаях, когда у вас есть индикатор выполнения процесса требующей значительного времени операции, и вы хотите дать пользователю возможность отменить ее.

Диалоговые окна

- Поскольку у такого окна нет собственного цикла событий, для его обновления вы должны периодически вызывать метод QApplication::processEvents.
- **QDialog** — базовый класс диалоговых окон в Qt, предоставляющий методы **exec** и **show** для обработки модальных и немодальных диалоговых окон, у него есть встроенный класс QLayout, который можно использовать, и несколько сигналов и слотов, полезных для формирования откликов на нажатие кнопки.
- Обычно вы будете создавать для своих диалоговых окон класс-потомок QDialog и вставлять в него виджеты для создания интерфейса диалогового окна.

Диалоговые окна

- В отличие от виджета типа QMainWindow вы можете задать объект MyDialog как родительский для своего объекта QLayout без создания пустого QWidget в качестве родительского.
- У объекта QDialog есть два слота — **accept** и **reject**, которые применяются для обозначения результата. Этот результат возвращается методом **exec**. Как правило, вы будете связывать кнопки **OK** и **Cancel** со слотами, как в MyDialog.
- **Модальные диалоговые окна**
- Для применения диалогового окна как модального вы вызываете метод exec, который открывает диалоговое окно и возвращает QDialog::Accepted или QDialog::Rejected в зависимости от того, какой слот был активизирован:

Диалоговые окна

```
MyDialog* dialog = new MyDialog(this, "mydialog");
if (dialog->exec() == QDialog::Accepted) {
 // Пользователь щелкнул мышью кнопку OK
 doSomething();
} else {
 // Пользователь щелкнул мышью кнопку Cancel или
 // диалоговое окно уничтожено
 doSomethingElse();
}
delete dialog;
```

Диалоговые окна

- Когда метод **exec** возвращает управление приложению, диалоговое окно автоматически скрывается, но вы все равно удаляете объект из памяти.
- Учтите, что когда вызывается exec, вся обработка прекращается.
- **Немодальные диалоговые окна**
- Немодальные диалоговые окна слегка отличаются от обычных основных окон прежде всего тем, что располагаются поверх своего родительского окна, совместно используют их элемент на панели задач и автоматически скрываются, когда вызван слот accept или reject.

Диалоговые окна

- Для отображения немодального диалогового окна вызывайте метод show, как вы сделали бы для окна QMainWindow:
- MyDialog *dialog = new MyDialog(this, "mydialog");
- dialog->show();
- Функция **show** выводит диалоговое окно на экран и немедленно возвращается в приложение для продолжения цикла выполнения. Для обработки нажатой кнопки вы должны написать слоты и установить с ними связь.
- Как и в случае модального окна, диалоговое окно автоматически скрывается при нажатии кнопки.

Диалоговые окна

- **Полумодальное диалоговое окно**
- Для создания полумодального диалогового окна вы должны задать флаг модального режима в конструкторе QDialog и применить метод **show**:
- **QDialog(QWidget *parent=0, const char *name=0, bool modal=FALSE, WFlags f=0);**
- Вы не задаете в модальном диалоговом окне флаг модального режима равным TRUE, потому что вызов exec заставляет диалоговое окно перейти в модальный режим независимо от значения этого флага.
- Конструктор вашего диалогового окна будет выглядеть примерно следующим образом:

Диалоговые окна

- MySMDialog::MySMDialog(QWidget *parent, const char *name):
- QDialog(parent, name, TRUE) {
- ...
- }
- После того как вы определили ваше диалоговое окно, вызовите функцию **show** и затем продолжите свою обработку, периодически вызывая QApplication::processEvents для обновления вашего диалогового окна:
- MySMDialog *dialog = MySMDialog(this, "semimodal");
- dialog->show();

Диалоговые окна

- while (processing) {
- doSomeProcessing();
- app->processEvents();
- if (dialog->wasCancelled()) break;
- }
- Перед продолжением выполнения проверьте, не уничтожено ли диалоговое окно. Функция wasCancelled не является частью класса QDialog — вы должны написать ее самостоятельно.
- Комплект Qt предоставляет готовые подклассы класса QDialog, предназначенные для конкретных задач, таких как выбор файлов, ввод текста, индикация процесса выполнения и вывод окна сообщения.

Диалоговые окна

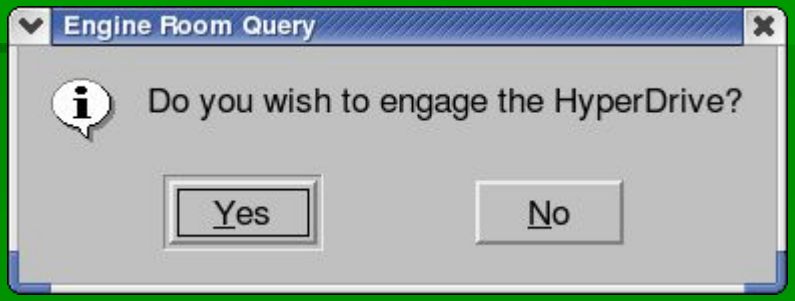
- QMessageBox — модальное диалоговое окно, отображающее простое сообщение с пиктограммой и кнопками. У класса QMessageBox есть статические методы создания и отображения окон всех трех перечисленных типов:
- **#include <qmessagebox.h>**
- **int information(QWidget *parent, const QString& caption, const QString&text, int button0, int button1=0, int button2=0);**
- **int warning(QWidget *parent, const QString& caption, const QString& text, int button0, int button1, int button2=0);**

Диалоговые окна

- **int critical(QWidget *parent, const QString& caption, const QString& text, int button0, int button1, int button2=0);**
- Можно выбрать кнопки из списка готовых кнопок QMessageBox, соответствующих значениям, возвращаемым статическими методами:
- QMessageBox::Ok;
- QMessageBox::Cancel;
- QMessageBox::Yes;
- QMessageBox::No;
- QMessageBox::Abort;
- QMessageBox::Retry;
- QMessageBox::Ignore.

Диалоговые окна

- Типичный пример использования окна QMessageBox будет похож на содержимое файла MessageBox.cpp.
- Вы соединили операцией OR (|) коды кнопок с вариантами Default и Escape, чтобы задать стандартные действия, при нажатии клавиш <Enter> (или <Return>) и <Esc>. Результирующее диалоговое окно показано на рис.:



Диалоговые окна

- Окно **QInputDialog** полезно для ввода пользователем отдельных значений, будь то текст, вариант раскрывающегося списка, целочисленное или действительное значение. У класса QInputDialog есть статические методы, например QMessageBox, создающие некоторые проблемы, поскольку у них слишком много параметров:
- **#include <qinputdialog.h>**
- **QString getText(const QString& caption, const QString& label, QLineEdit::EchoMode mode=QLineEdit::Normal, const QString& text=QString::null, bool* ok = 0, QWidget* parent = 0, const char * name = 0);**

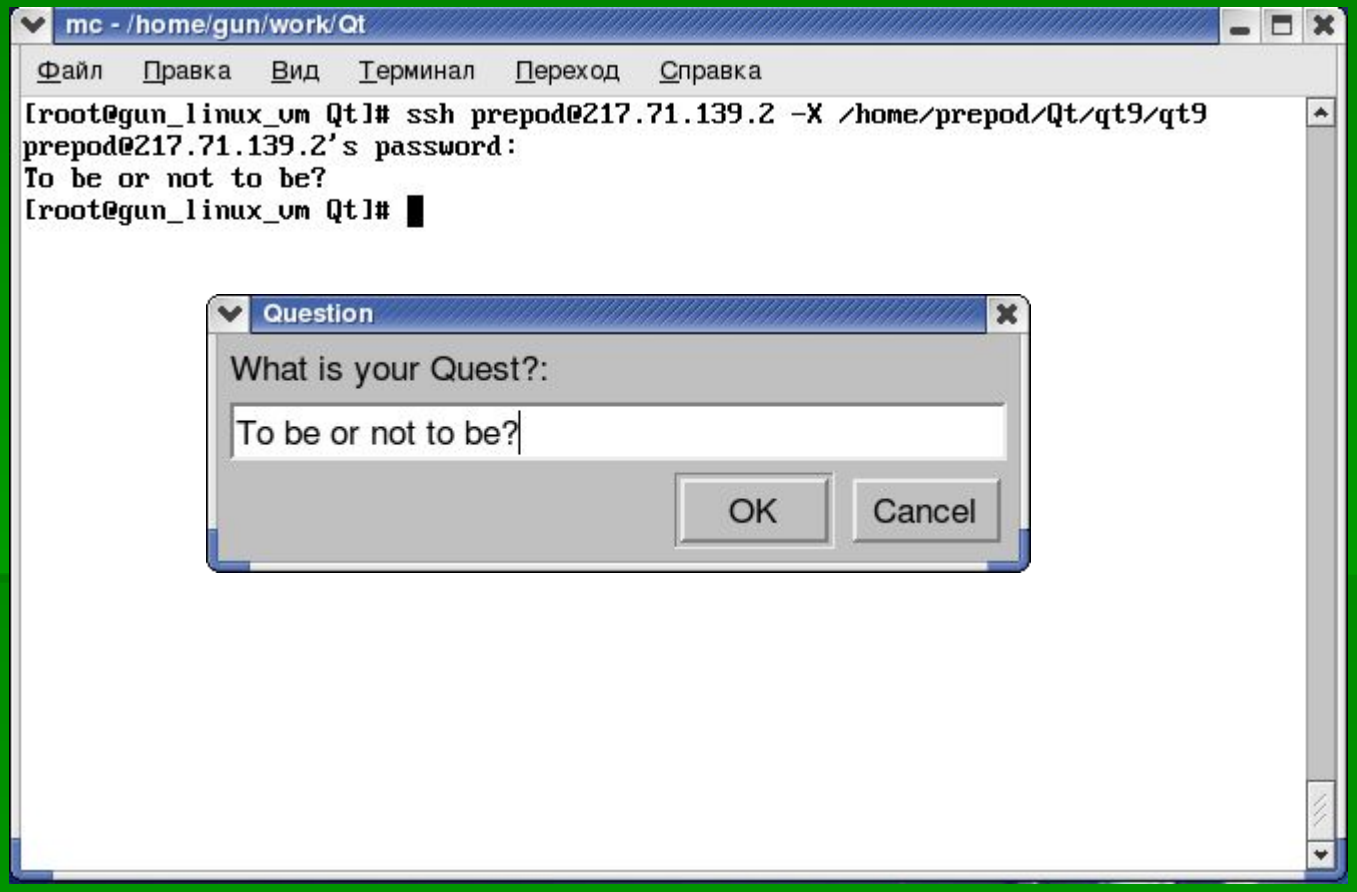
Диалоговые окна

- QString getItem(const QString& caption, const QString& label, const QStringList& list, int current=0, bool editable=TRUE, bool* ok=0, QWidget* parent = 0, const char* name=0)
- int getInteger(const QString& caption, const QString& label, int num=0, int from = -2147483647, int to = 2147483647, int step = 1, bool* ok = 0, QWidget* parent = 0, const char* name = 0);
- double getDouble(const QString& caption, const QString& label, double num = 0, double from = -2147483647, double to = 2147483647, int decimals = 1, bool* ok = 0, QWidget* parent = 0, const char* name = 0);

Диалоговые окна

- Для ввода строки текста применяют фрагмент кода:
- bool result;
- QString text = QInputDialog::getText("Question", "What is your Quest?:", QLineEdit::Normal, QString::null, &result, this, "input");
- if (result) {
- doSomething(text);
- } else {
- // Пользователь нажал Cancel
- }
- Как видно из рис., окно QInputDialog создано с помощью виджета QLineEdit и кнопок **OK** и **Cancel**.

Диалоговые окна



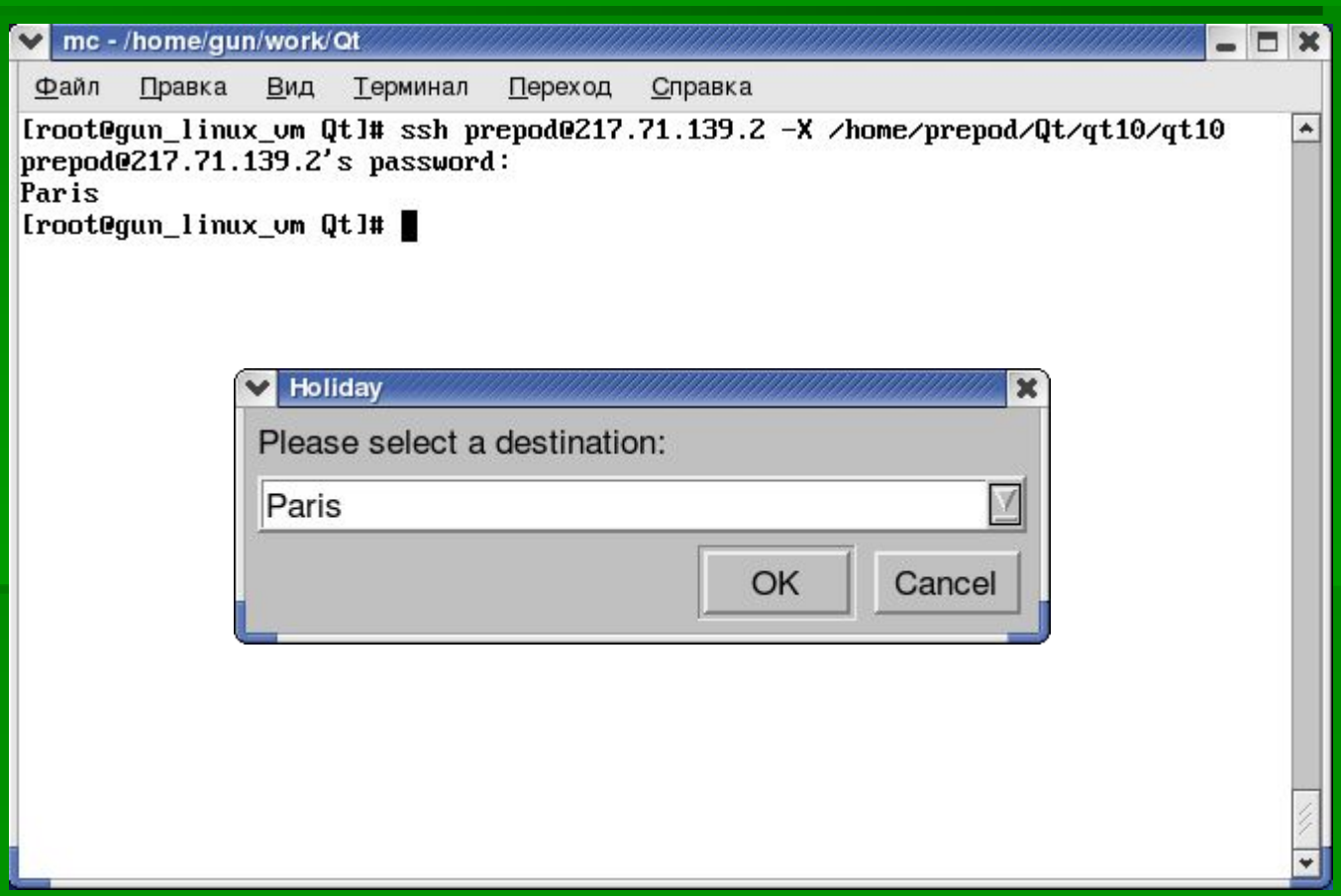
Диалоговые окна

- Диалоговое окно, созданное методом QInputDialog::getText, применяет виджет QLineEdit. Параметр режима редактирования, передаваемый в функцию getText , управляет способом отображения набираемого текста точно так же, как аналогичный параметр режима виджета QLineEdit. Вы можете также задать текст, выводимый по умолчанию, или оставить поле пустым, как показано на рис. 9. У всех окон QInputDialog есть кнопки **OK** и **Cancel**, и в метод передается указатель типа bool для обозначения нажатой кнопки — результат равен TRUE, если пользователь щелкает мышью кнопку **OK**.

Диалоговые окна

- Метод **getItem** с помощью раскрывающегося списка QComboBox предлагает пользователю список вариантов:
- bool result;
- QStringList options;
- options << "London" << "New York" << "Paris";
- QString city = QInputDialog::getItem("Holiday", "Please select a
- destination:", options, 1, TRUE, &result, this, "combo");
- if (result) selectDestination(city);
- Созданное диалоговое окно показано на рис.

Диалоговые окна



Диалоговые окна

- Функции **getInteger** и **getDouble** действуют во многом аналогично, поэтому мы не будем на них останавливаться.
- Список литературы:
- 1. Н. Мэтью, Р. Стоунс. Основы программирования в Linux. БХВ-Петербург, 2009 г. - 896 стр.
- 2. Qt 4.1: Qt Справочная Документация (Desktop Edition) [Электронный ресурс] // Qt 4.1 Русскоязычная документация: [сайт]. URL: http://qtdocs.narod.ru (дата обращения: 14.08.2013).