Making GUIs with Qt 4

Besides making clickable programs, learning to program GUIs will give you several other skills with C++:

- Event-based programming
- Working with a (very) large library
- Managing memory in more complicated programs

Getting Started

```
#include <QtGui>

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

    QLabel hello("Hello World!");

    hello.resize(250, 150);
    hello.setWindowTitle("Simple example");
    hello.show();

    return app.exec();
}
```

Building Qt Applications

- Qt has its own preprocessor, the Meta Object Compiler (moc)
- `qmake` manages Qt projects and generates makefiles automatically
  - `qmake -project` will make a project file (ends in .pro) that configures the makefile
  - `qmake` makes a makefile
- So, to build a Qt project: `qmake -project; qmake; make`

Qt Overview

- There is one, and only one, QApplication
- `qApp` is a global pointer to the QApplication

Qt Overview

- There is one, and only one, QApplication
- `qApp` is a global pointer to the QApplication
- Everything clickable is called a 'widget'
- Widgets can hold other widgets
- A widget with no parent becomes a window

A Simple Notepad

```
#include<QApplication>
#include<QTextEdit>

int main(int argc, char** argv)
{
  QApplication app(argc,argv);

  QTextEdit te;
  te.setWindowTitle("Not Vim");
  te.show();

  return app.exec();
}
```

Composite Objects

- Widgets can be added to another widget with the `addWidget()` function
- You can use a Layout to specify how the widgets are organized

Composite Objects

- Widgets can be added to another widget with the `addWidget()` function
- You can use a Layout to specify how the widgets are organized
- Memory Management: `addWidget()` takes a pointer and is responsible for cleaning up all its children

Layout Example

```
#include<QtGui>

int main(int argc, char** argv)
{
  QApplication app(argc,argv);

  QTextEdit* te = new QTextEdit;
  QPushButton* quit = new QPushButton("&Quit");

  QVBoxLayout* layout = new QVBoxLayout;
  layout->addWidget(quit);
  layout->addWidget(te);

  QWidget window;
  window.setLayout(layout);

  window.show();
```

Making Buttons Do Things

- Qt is event-driven: QApplication monitors what the user does and sends events to widgets when something happens
- Signal: An event a widget causes: button click, key press, etc.
- Slot: An action a widget takes when a signal is sent
- `connect(source-object, SIGNAL(signal_name()), destination-object, SLOT(slot_name()))` connects signals to slots

Actually Quitting

```
#include<QtGui>
int main(int argc, char** argv)
{
  QApplication app(argc,argv);

  QTextEdit* te = new QTextEdit;
  QPushButton* quit = new QPushButton("&Quit");

  QObject::connect(quit, SIGNAL(clicked()),
      qApp, SLOT(quit()));

  QVBoxLayout* layout = new QVBoxLayout;
  layout->addWidget(quit);
  layout->addWidget(te);

  QWidget window;
  window.setLayout(layout);
```

Writing your own slot

- In order to make your own slots, you need to make a custom `QWidget` class
- In addition to public and private functions and members, QObjects have public and private slots
- A slot is just a function that gets called whenever a signal connected to it is sent

Example: `ask-quit`

Menus and Toolbars

- `QMainWindow` is a class for making standard applications with menus and toolbars
- `setCentralWidget()` sets the widget that fills the window
- `menuBar()` returns a pointer to the menubar, which you can use to add new menus
- `addToolbar()` creates a new toolbar

Menus and Toolbars

- `QMainWindow` is a class for making standard applications with menus and toolbars
- `setCentralWidget()` sets the widget that fills the window
- `menuBar()` returns a pointer to the menubar, which you can use to add new menus
- `addToolbar()` creates a new toolbar
- To avoid repeating a lot of code, you can add a `QAction` to both a menu and a toolbar
- Then you can connect that one action to various slots

Example: `menus`

Getting Fancy with Signals

- You can declare your own signals in the `signals:` section
- For example, `QPushButton` has `void clicked();` in its signals

Getting Fancy with Signals

- You can declare your own signals in the `signals:` section
- For example, `QPushButton` has `void clicked();` in its signals
- To send a signal, use `emit signal-name()`

Getting Fancy with Signals

- You can declare your own signals in the `signals:` section
- For example, `QPushButton` has `void clicked();` in its signals
- To send a signal, use `emit signal-name()`
- You don't actually implement signals, just declare, emit, and connect them

Getting Fancy with Signals

- You can declare your own signals in the `signals:` section
- For example, `QPushButton` has `void clicked();` in its signals
- To send a signal, use `emit signal-name()`
- You don't actually implement signals, just declare, emit, and connect them
- You can send data over signals by adding parameters to your signals!
- Connect that signal to a slot that takes the same arguments
- The slot will be called with the data you use when you emit the signal

Example: `title`