More Qt!
A very simple program
(part 1 of 9) - Full source code

// simple.C
//
// A very simple Qt program - just a single button that
// closes the program when pressed.

#include <qapp.h>
#include <qpushbutton.h>

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

    QPushButton myButton( "Quit", NULL );
    QObject::connect( &myButton, SIGNAL(clicked()), &a, SLOT(quit()) );

    myButton.show();
    a.setMainWidget( &myButton );

    return a.exec();
}
A very simple program
(part 2 of 9) - Header files

#include <qapp.h>
#include <qpushbutton.h>

- Includes the header files for QApplication and QPushButton
- Look at the Qt reference page for a class to see what header file is needed
A very simple program
(part 3 of 9) - The QApplication

QApplication a( argc, argv );

• Every Qt program has a QApplication
• The QApplication must be created before doing anything else
• Constructor needs the commandline arguments
  – Will process some of them, like XWindow system options; these will be removed from the list
  – Anything it doesn't understand will be left alone
A very simple program
(part 4 of 9) - Creating a QPushButton

QPushButton myButton( "Quit", NULL );

• QPushButton constructor takes text to appear on button and a pointer to the parent widget
  - Every widget takes a pointer to a parent widget
  - This is the only widget in the program, so there is no parent, so NULL is used.
  - Passing NULL as the parent means this is a top-level widget (it has its own window)

• There are other constructors for the QPushButton class; refer to the Qt documentation for details.
A very simple program
(part 5 of 9) - Connecting things

QObject::connect( &myButton, SIGNAL(clicked( )), &a, SLOT(quit( )) );

• The **clicked( )** signal of the QPUSHButton is emitted when the button is pressed

• Signalling the **quit( )** slot of the QApplication will cause the program to terminate

• Connecting the two means the pressing the button will make the program quit
myButton.show();

• When a widget is created, it is invisible
• Must call `show()` to make it visible
  – Won't actually be shown until calling the `exec()` function in the QApplication
• Calling `show()` on one widget will make all children visible
  – If you create a child widget after calling `show()` on the parent, the child will still be invisible
A very simple program
(part 7 of 9) - setMainWidget

```cpp
a.setMainWidget(&myButton);
```

- When the “main widget” is destroyed, the application terminates
- Should be there, though usually the application is terminated by signalling `quit()` in the QApplication object
A very simple program
(part 8 of 9) - a.exec( )

return a.exec( );

• Displays any visible widgets and starts the event processing loop
• Returns when the application is terminated
A very simple program
(part 9 of 9) - Compiling and running the program

• Compile and run with:

  > g++ -c simple.C -I/usr/lib/qt-3.1/include
  > g++ -o simple simple.o -L/usr/lib/qt-3.1/lib -lqt-mt
  > ./simple

• What you get:
Layout Widgets
(part 1 of 5) What are they?

- A layout widget arranges its children in some way
  - QVBox stacks its children vertically
  - QHBox lines them up horizontally
  - QGrid arranges them in a grid, filling up one row/column before starting another row/column
Layout Widgets
(part 2 of 5) Example program source code

// layout.C
//
// A slightly more complex Qt program that uses a QVBox to arrange
// a couple of child widgets

#include <qapp.h>
#include <qpushbutton.h>
#include <qlabel.h>
#include <qvbox.h>

int main( int argc, char ** argv )
{
    QApplication a( argc, argv );
    QVBox box;
    QLabel hello( "Hello World!", &box );
    QPushButton quit( "Quit", &box );

    QObject::connect( &quit, SIGNAL(clicked( )), &a, SLOT(quit(  )) );

    box.show( );
    a.setMainWidget( &box );
    return a.exec( );}
Creating the QVBox

QVBox box;

• Creates a QVBox widget
• The QVBox constructor (as well as those for other layout widgets) can take a pointer to a parent widget
• If no pointer is passed in, the constructor assumes there is no parent (it uses a default value)
Layout Widgets
(part 4 of 5) Adding children and making everything visible

QLabel hello( "Hello World!", &box );
QPushButton quit( "Quit", &box );

• Create a label and push button with the QVBox as the parent

• Generally, child widgets are drawn inside the parent widget

box.show( );

• Calling show on the QVBox will make both it and its children visible
Layout Widgets
(part 5 of 5) What you get...

• Compile and run the program to get...
A Longer Example
(part 1 of 3) Some of the source code

// numbers.C
//
// A simple Qt program that actually does something.

#include <qapp.h>
#include <qpushbutton.h>
#include <qslider.h>
#include <qlcdnumber.h>
#include <qvbox.h>

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

    // A QVBox will stack its child widgets vertically
    QVBox box ;

    // Create an LCD number that displays in binary
    QLCDNumber bin( &box ) ;
    bin.setMode( QLCDNumber::Bin ) ;
    bin.setNumDigits( 8 ) ;
    bin.setSegmentStyle( QLCDNumber::Filled ) ;
A Longer Example
(part 2 of 3) The rest of the source code

// Create an LCD number that displays in decimal
QLCDNumber dec( &box );
dec.setSegmentStyle( QLCDNumber::Filled );

// Create a slider to pick a number
QSlider pickNum( Qt::Horizontal, &box );
pickNum.setRange( 0, 255 );

// Connect the slider to the LCDs
QObject::connect( &pickNum, SIGNAL(valueChanged(int)),
                   &dec, SLOT(display(int)) );
QObject::connect( &pickNum, SIGNAL(valueChanged(int)),
                   &bin, SLOT(display(int)) );

// Create and connect quit button
QPushButton quit( "Quit", &box );
QObject::connect( &quit, SIGNAL(clicked( )), &a, SLOT(quit( )) );

box.show( );
a.setMainWidget( &box );
return a.exec( );
A Longer Example
(part 3 of 3) What you get...

• A program for converting between binary and decimal numbers:
Making New Widgets
(part 1 of 4) Class definition

• Existing widgets can be used as a starting point...

// converter.h

#ifdef CONVERTER_H
#define CONVERTER_H

#include <qvbox.h>

// Inherit from QVBox to take advantage of its layout
// and resizing functionality
class Converter : public QVBox{

public:

    // Widget constructors must always have parameters for parent widget
    // and name. The widget name is used for debugging.
    Converter( QWidget * parent = 0, const char * name = 0 );

};

#endif
Making New Widgets
(part 2 of 4) Constructor implementation

// converter.C

#include <qaccel.h>
#include <qlcdnumber.h>
#include <qslider.h>
#include "converter.h"

Converter::Converter( QWidget * parent, const char * name )
    : QVBox( parent, name )  // Pass parent widget and widget name to
      // parent constructor
{
    // Create an LCD number that displays in binary with this
    // widget as the parent
    QLCDNumber * bin = new QLCDNumber( this );
    bin->setMode( QLCDNumber::Bin );
    bin->setNumDigits( 8 );

    // Makes the LCD number easier to read
    bin->setSegmentStyle( QLCDNumber::Filled );
Making New Widgets
(part 3 of 4) Constructor implementation continued

// Create an LCD number that displays in decimal
QLCDNumber * dec = new QLCDNumber( this );
dec->setSegmentStyle( QLCDNumber::Filled );

// Create a slider to pick a number
QSlider * pickNum = new QSlider( Qt::Horizontal, this );
pickNum->setRange( 0, 255 );
pickNum->setSteps( 1, 1 );

// Connect the slider to the LCDs
connect( pickNum, SIGNAL(valueChanged(int)),
         dec, SLOT(display(int)) );
connect( pickNum, SIGNAL(valueChanged(int)),
         bin, SLOT(display(int)) );

} // end of constructor

• Note that the pointers to the child widgets are local variables – Qt will keep track of them and destroy them when this widget is destroyed
// main.C
//
// Main function for conversion
// widget program

#include <qapp.h>
#include "converter.h"

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

    // Create a converter widget
    Converter convert;

    // Make the widget bigger
    convert.resize( 400, 400 );

    converter.show( );
    a.setMainWidget( &converter );
    return a.exec( );
}
Making Really New Widgets

- Inherit from QWidget
- Implement inherited functions
  - `paintEvent` to draw the widget
  - `mousePressEvent`, `mouseMoveEvent`, and `mouseReleaseEvent` to handle mouse interaction
  - `sizeHint` and `sizePolicy` to tell Qt how your widget likes to be sized