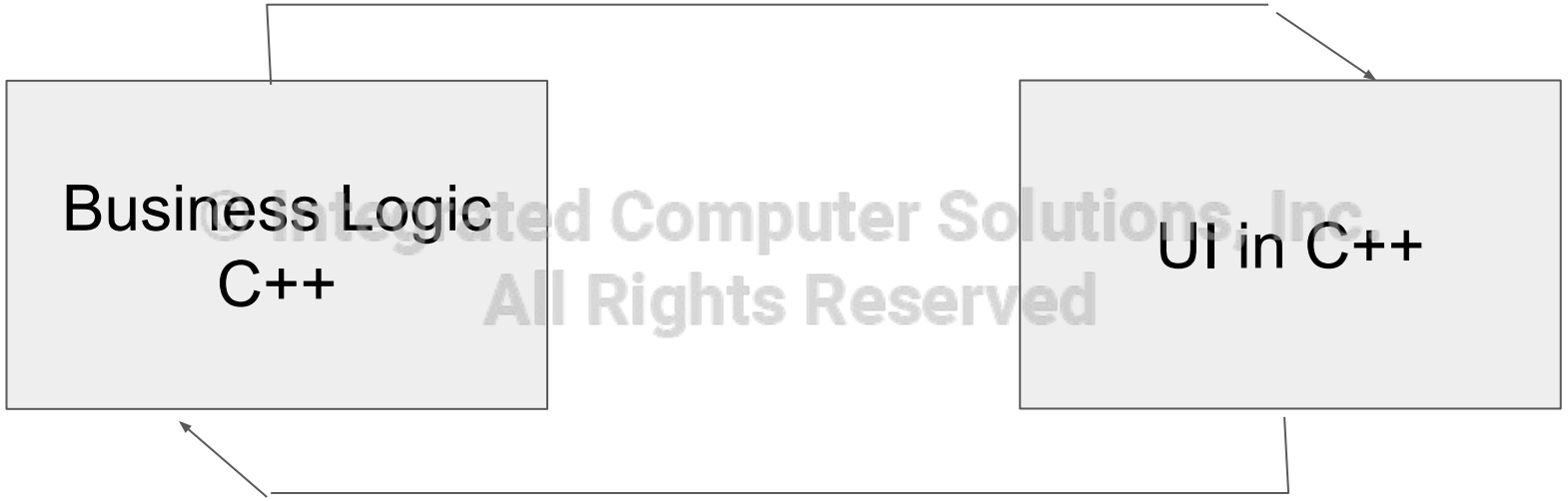
Qt For Beginners Training

Part 4

Business Logic/UI Paradigm

-



Widgets vs QML

The QWidget API is in C++, compiled and more suitable for Desktop Applications

- Adapts to the style of OS/Platform running the application
- Layouts makes seamlessly resizing applications easier
- Desktop platform specific standards are built-in
- Easy handling of Dialogs through QDialog

QML is a markup language which uses Javascript and is interpreted at run -time. More suitable for embedded devices.

- More Control

- Branding User Interfaces are easier

- Touch screen specific features readily available: rotating screens easier, multitouch, sliding screens.

A First Example with Widgets

```
// Container (window) widget creation
QWidget container; // on stack, or managed by another object
QLabel *label = new QLabel("Note:", &container);
QTextEdit *edit = new QTextEdit(&container);
QPushButton *clear = new QPushButton("Clear", &container);
QPushButton *save = new QPushButton("Save", &container);

// Widget layout
QVBoxLayout *outer = new QVBoxLayout();
outer->addWidget(label);
outer->addWidget(edit);
QHBoxLayout* inner = new QHBoxLayout();
inner->addWidget(clear);
inner->addWidget(save);

container.setLayout(outer);
outer->addLayout(inner);  // Nesting layouts
```

Gallery of Widgets

- **QLabel**

  label = new QLabel("Text", parent);

  setPixmap(pixmap) - as content

- **QLineEdit**

```
line = new QLineEdit(parent);
line->setText("Edit me");
line->setEchoMode(QLineEdit::Password);
connect(line, SIGNAL(textChanged(QString)) …
setInputMask(mask)
setValidator(validator)
```

- **QTextEdit**

```
edit = new QTextEdit(parent);
edit->setPlainText("Plain Text");
edit->append("<h1>Html Text</h1>");
connect(edit, SIGNAL(textChanged(QString)) ...
```

Gallery of Widgets: Button Widgets

- **QAbstractButton**
  - Abstract base class of buttons
- **QPushButton**

```
button = new QPushButton("Push Me", parent);
button->setIcon(QIcon("images/icon.png")); connect
SIGNAL(clicked()) …
```

`setCheckable(bool)` - toggle button

- **QRadioButton**

```
radio = new QRadioButton("Option 1", parent);
```

Radio buttons are `autoExclusive` by default

- **QCheckBox**

```
check = new QCheckBox("Choice 1", parent);
```

Button Widgets (continued)

- **QToolButton**

```
button = new QToolButton(parent);

button->setDefaultAction(action);

button->setToolButtonStyle(Qt::ToolButtonTextUnderIcon);
```


- **QButtonGroup** - non-visual/non-widget button manager

```
group = new QButtonGroup(parent);

group->addButton(button); // add more buttons

group->setExclusive(true);

connect(group, SIGNAL(buttonClicked(QAbstractButton*)) ...
```

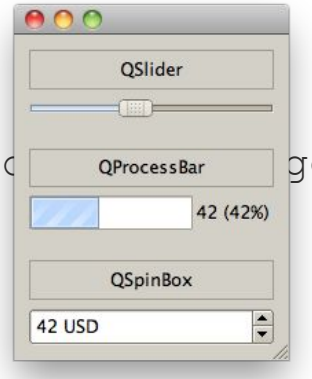
Value Widgets

- **QSlider**

```
slider = new QSlider(Qt::Horizontal, parent); slic                ge(0,
99);

slider->setValue(42);

connect(slider, SIGNAL(valueChanged(int)) ...
```



- **QProgressBar**

```
progress = new QProgressBar(parent);
progress->setRange(0, 99);

progress->setValue(42);

// format: %v for value; %p for percentage
progress->setFormat("%v (%p%)");
```

Value Widgets (continued)

- **QSpinBox**

```
spin = new QSpinBox(parent);
spin->setRange(0, 123); // The default is 0 - 99
spin->setValue(42);
spin->setSuffix(" USD");
connect(spin, SIGNAL(valueChanged(int)) …
```


- **QDoubleSpinBox**

```
dspin = new QDoubleSpinBox(parent);
dspin->setRange(0.0, 1.0);
dspin->setValue(0.5);
dspin->setSuffix(" Kg");
connect(spin, SIGNAL(valueChanged(double)) ...
```

Organizer Widgets

- **QGroupBox**

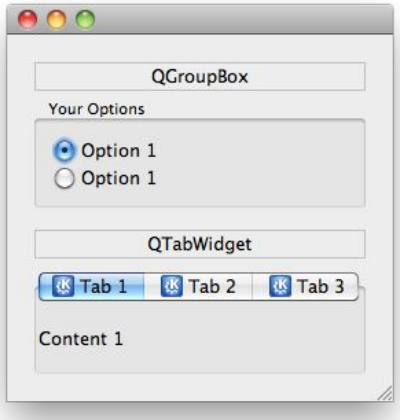
```
box = new QGroupBox("Your Options", parent);
// ... set layout and add widgets
```

`setCheckable(bool)` - checkbox in title

- **QTabWidget**

```
tab = new QTabWidget(parent);
tab->addWidget(widget, icon, "Tab 1");
connect(tab, SIGNAL(currentChanged( int)) …
```

- `setCurrentWidget(widget)`
  - Displays page assoziated by widget
- `setTabPosition(position)`
  - Defines where tabs are drawn
- `setTabsClosable(bool)`
  - Adds close buttons



Creating a Custom Widget

- It's as easy as deriving from QWidget

```
class CustomWidget : public QWidget
{
Public:
    explicit CustomWidget(QWidget* parent=0);
}
```


- If you need custom signals, slots, or properties
  - add Q_OBJECT
- Use child widget members (composition)
- ...or paint the widget yourself (from scratch)

Painting on Widgets

- Override `paintEvent(QPaintEvent*)`

```
void CustomWidget::paintEvent(QPaintEvent *) {
    QPainter painter(this);
    painter.drawRect(0,0,100,200); // x,y,w,h
}
```

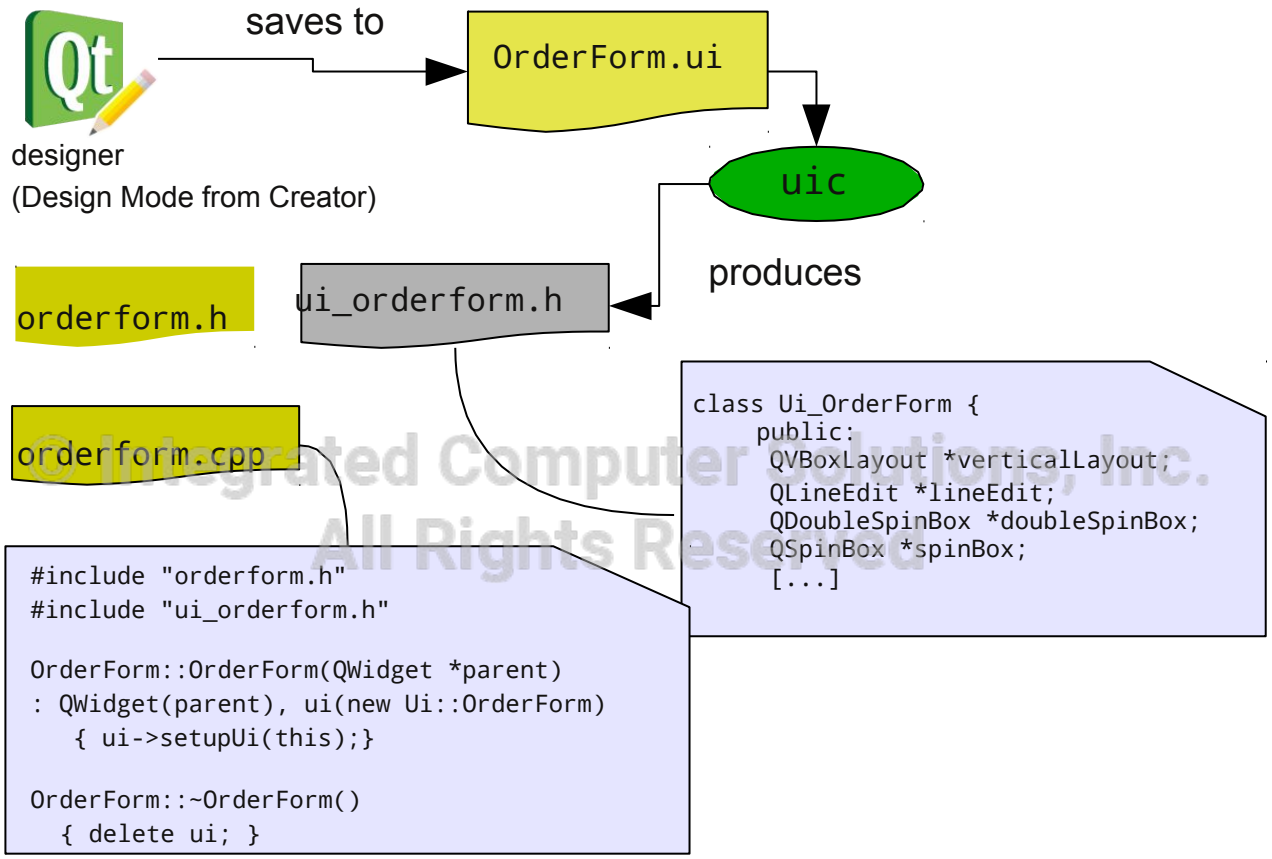

- Schedule painting
  - `update()`: schedules paint event
  - `repaint()`: repaints directly (*not recommended*)

Re-Implementing Event Handlers

- **Overload needed event handlers**
  - Often:
    - `QWidget::mousePressEvent()`,

      `QWidget::mouseReleaseEvent()`
  - If widget accepts keyboard input
    - `QWidget::keyPressEvent()`
  - If widget changes appearance on focus
    - `QWidget::focusInEvent()`,
    - `QWidget::focusOutEvent()`

From .ui to C++



In the Next Session

- Model/View