Qt Essentials - Objects Module
Training Course

Visit us at http://qt.digia.com

Produced by Digia Plc.

Material based on Qt 5.0, created on September 27, 2012

Digia Plc.
Module: Object Communication

- Signals & Slots
- Event Handling
Module Learning Objectives

• Learn ...
  • ... how objects communicate
  • ... details of signals & slots
  • ... which variations for signal/slot connections exist
  • ... how to create custom signals & slots
  • ... what the role of the Qt event loop is
  • ... how Qt handles events
Object Communication

- **Between objects**
  Signals & Slots

- **Between Qt and the application**
  Events

- **Between Objects on threads**
  Signal & Slots + Events

- **Between Applications**
  DBus, QSharedMemory
Module: Object Communication

- Signals & Slots
  - Event Handling
General Problem

How do you get from "the user clicks a button" to your business logic?

- Possible solutions
  - Callbacks
    - Based on function pointers
    - Not type-safe
  - Observer Pattern (Listener)
    - Based on interface classes
    - Needs listener registration
    - Many interface classes

- Qt uses
  - Signals and slots for high-level (semantic) callbacks
  - Virtual methods for low-level (syntactic) events.
Connecting Signals to Slots

Signals & Slots

Object Communication
Connecting Signals to Slots

Signals & Slots

Signal emitted

42
Connecting Signals to Slots

Signals & Slots

Object Communication
Connecting Signals to Slots

Signal/Slot connection
Connecting Signals to Slots

QObject::connect(slider, &QSlider::valueChanged, spinbox, &QSpinBox::setValue)
void QSlider::mousePressEvent(...) {
    ...
    emit valueChanged( newValue );
    ...
}
Connecting Signals to Slots

```cpp
void QSpinBox::setValue( int value )
{
    ...
    m_value = value;
    ...
}
```

Signals & Slots
Connecting Signals to Slots

```cpp
void QSlider::mousePressEvent(...) {
    ...
    emit valueChanged( newValue );
    ...
}
```

```cpp
void QSpinBox::setValue( int value ) {
    ...
    m_value = value;
    ...
}
```

Signal emitted

Signal/Slot connection

Slot implemented

```cpp
QObject::connect( slider, &QSlider::valueChanged, spinbox, &QSpinBox::setValue )
```
Connection variants

- **Qt 4 style:**
  
  ```cpp
  connect( slider, SIGNAL(valueChanged(int)),
          spinbox, SLOT(setValue(int)));
  ```

- **Using function pointers:**
  
  ```cpp
  connect( slider, &QSlider::valueChanged,
          spinbox, &QSpinBox::setValue );
  ```

- **Using non-member function:**
  
  ```cpp
  static void printValue(int value) {...}
  connect( slider, &QSignal::valueChanged, &printValue );
  ```

- **Using C++11 lambda functions:**
  
  ```cpp
  connect( slider, &QSlider::valueChanged,
           [=] (int value) {...} );
  ```
connect - function pointers

- **Qt 5 components**
  ```cpp
cconnect( slider, &QSlider::valueChanged, 
             spinbox, &QSpinBox::setValue );
```

- **Primary choice when connecting objects**
  - ✓ Compile time errors
  - ✓ No special syntax for slots
  - ✓ Q_OBJECT not need for slots
  - ✗ connecting to overloaded slots is hard

Demo object-communication/ex-connect-function-pointers
• Qt 4 ported code:

```cpp
connect( slider, SIGNAL(valueChanged(int)),
        spinbox, SLOT(setValue(int)));
```

Receiving object:

X need to declare the slot in a slots section
X need the Q_OBJECT macro
X need to have moc run on it

X Only run time errors
✓ overloaded slot are easy
✓ Existing Qt4 code do not need to be rewritten
• **File: myclass.h**

```cpp
class MyClass : public QObject {
    Q_OBJECT // marker for moc
    // ...
public slots:
    void setValue(int value); // a custom slot
};
```

• **File: myclass.cpp**

```cpp
void MyClass::setValue(int value) {
    // slot implementation
}
```
Using non-member functions:

```cpp
static void printValue(int value) {
    qDebug( "value = %d", value );
}
connect( slider, &QSignal::valueChanged, &printValue );
```

- No slots syntax, no Q_OBJECT, no moc
- Compile time errors
- Any function, e.g. the return value of std::bind
Using C++11 lambda functions:

```cpp
connect( slider, &QSlider::valueChanged,
         [=] (int value) { qDebug("%d", value); } );
```

- No slots syntax, no Q_OBJECT, no moc
- Compile time errors
- No need for an extra function
Custom Signals

- **File: myclass.h**
  ```cpp
class MyClass : public QObject {
    Q_OBJECT // marker for moc
    // ...
    signals:
    void valueChanged(int value); // a custom signal
};
```

- **File: myclass.cpp**
  ```cpp
  // No implementation for a signal
  ```

- **Sending a signal**
  ```cpp
  emit valueChanged(value);
  ```
Q_OBJEKT - flag for MOC

- **Q_OBJEKT**
  - Enhances QObject with meta-object information
  - Required for signals
  - Required for slots when using the Qt4 way

- **moc** creates meta-object information
  
  ```
  moc -o moc_myclass.cpp myclass.h
  c++ -c myclass.cpp; c++ -c moc_myclass.cpp
  c++ -o myapp moc_myclass.o myclass.o
  ```

- **qmake** takes care of mocing files for you
Variations of Signal/Slot Connections

<table>
<thead>
<tr>
<th>Signal(s)</th>
<th>Connect to</th>
<th>Slot(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>√</td>
<td>many</td>
</tr>
<tr>
<td>many</td>
<td>√</td>
<td>one</td>
</tr>
<tr>
<td>one</td>
<td>√</td>
<td>another signal</td>
</tr>
</tbody>
</table>

- **Signal to Signal connection**

  ```cpp
  connect(bt, SIGNAL(clicked()), this, SIGNAL(okSignal()));
  ```

- **Not allowed to name parameters**

  ```cpp
  connect(m_slider, SIGNAL(valueChanged(int value)),
  this, SLOT(setValue(int newValue)))
  ```
## Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

<table>
<thead>
<tr>
<th>Signal</th>
<th>Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>rangeChanged(int,int)</td>
<td>setRange(int,int)</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>valueChanged(int)</td>
<td>setValue(int)</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>textChanged(QString)</td>
<td>setRange(int,int)</td>
</tr>
<tr>
<td></td>
<td>†</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

† Though not for Qt4 connection types
• Create an application as shown here
  • Clicking on ``Select Color''
    updates label with color's name.

• Hints
  • QColorDialog::getColor() to fetch a color
  • QColor::name() to get the color name

• Optional
  • In QColorDialog, honor the user clicking ``cancel'', and provide it with
    the current color to start from.
  • Set the selected color as the label's background
    • Hint: see QPalette
    • Hint: see QWidget::setAutoFillBackground()
Lab: Source Compatibility

- **Implement custom slider**
  - API compatible with QSlider
  - Shows current value of slider

- **To create custom slider**
  - use QSlider and QLabel

- **To test slider**
  - main.cpp provides test code
  - QLCDNumber is part of test code

- **Optional:**
  - Discuss pros and cons of inheriting from QSlider instead of using an instance in a layout.
Module: Object Communication

- Signals & Slots
- Event Handling
Qt is an event-driven UI toolkit

QApplication::exec() runs the *event loop*

1. **Generate Events**
   - by input devices: keyboard, mouse, etc.
   - by Qt itself (e.g. timers)

2. **Queue Events**
   - by event loop

3. **Dispatch Events**
   - by QApplication to receiver: QObject
     - *Key events sent to widget with focus*
     - *Mouse events sent to widget under cursor*

4. **Handle Events**
   - by QObject event handler methods
Event Processing

Press Me

QQuickMouseArea

QQuickView

QPA

event loop

QCoreApplication

QQuickView

QQuickMouseArea

Object Communication
Event Processing

QPA

event loop

QCoreApplication

sendSpontaneousEvent
(QObject *, QEvent *)
Event Processing

QCoreApplication
1  event (QEvent *)
2  event (...)
3   mousePressEvent (...)

QWindow

QQuickCanvas

QQuickView
Event Processing

mousePressEvent (QMouseEvent *)

QWindow
QQuickCanvas
QQuickView
QQuickItem
QQuickMouseArea

Demo object-communication/ex-qml-event-backtrace

Event Handling
Event Handling

- `QObject::event(QEvent *event)`
  - Handles all events for this object

- Specialized event handlers for `QWidget` and `QQuickItem`:
  - `mousePressEvent()` for mouse clicks
  - `touchEvent()` for key presses

- Accepting an Event
  - `event->accept()` / `event->ignore()`
    - Accepts or ignores the event
    - Accepted is the default.

- Event propagation
  - Happens if event is ignored
  - Might be propagated to parent widget

Demo object-communication/ex-allevents
Example of Event Handling

- QCloseEvent delivered to top level widgets (windows)
- Accepting event allows window to close
- Ignoring event keeps window open

```cpp
void MyWidget::closeEvent(QCloseEvent *event) {
    if (maybeSave()) {
        writeSettings();
        event->accept(); // close window
    } else {
        event->ignore(); // keep window
    }
}
```
Multi threaded object communication

- Signal/slots between threads
- Posting events using
  ```
  QCoreApplication::postEvent(QObject* receiver, QEvent* event)
  ```
Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- How would you emit a signal?
- Can you return a value from a slot?
- When do you need to run qmake?
- Where do you place the Q_OBJECT macro and when do you need it?
- What is the purpose of the event loop?
- How does an event make it from the device to an object in Qt?
Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- How would you emit a signal?
- Can you return a value from a slot?
- When do you need to run qmake?
- Where do you place the Q_OBJECT macro and when do you need it?
- What is the purpose of the event loop?
- How does an event make it from the device to an object in Qt?
Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- **How would you emit a signal?**
- Can you return a value from a slot?
- When do you need to run qmake?
- Where do you place the Q_OBJECT macro and when do you need it?
- What is the purpose of the event loop
- How does an event make it from the device to an object in Qt?
Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- How would you emit a signal?
- Can you return a value from a slot?
- When do you need to run qmake?
- Where do you place the Q_OBJECT macro and when do you need it?
- What is the purpose of the event loop?
- How does an event make it from the device to an object in Qt?
Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- How would you emit a signal?
- Can you return a value from a slot?
- **When do you need to run qmake?**
- Where do you place the Q_OBJECT macro and when do you need it?
- What is the purpose of the event loop
- How does an event make it from the device to an object in Qt?
Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- How would you emit a signal?
- Can you return a value from a slot?
- When do you need to run qmake?

- **Where do you place the Q_OBJECT macro and when do you need it?**
- What is the purpose of the event loop
- How does an event make it from the device to an object in Qt?
Questions And Answers

• How do you connect a signal to a slot?
• How would you implement a slot?
• How would you emit a signal?
• Can you return a value from a slot?
• When do you need to run qmake?
• Where do you place the Q_OBJECT macro and when do you need it?
• **What is the purpose of the event loop**
• How does an event make it from the device to an object in Qt?
Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- How would you emit a signal?
- Can you return a value from a slot?
- When do you need to run qmake?
- Where do you place the Q_OBJECT macro and when do you need it?
- What is the purpose of the event loop
- How does an event make it from the device to an object in Qt?
© Digia Plc.

Digia, Qt and the Digia and Qt logos are the registered trademarks of Digia Plc. in Finland and other countries worldwide.