# 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

digia

# Module Learning Objectives

- Learn ...
  - ... how objects communication
  - ... 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

digia

- **Between objects**
  Signals & Slots
- **Between Qt and the application**
  Events
- **Between Objects on threads**
  Signal & Slots + Events
- **Between Applications**
  DBus, `QSharedMemory`

- Signals & Slots
- Event Handling

## General Problem

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

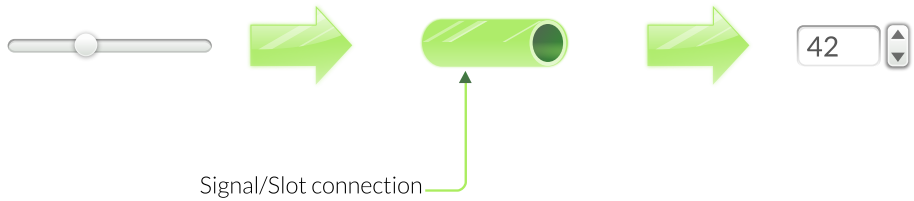
- Possible solutions
  - Callbacks
    - Based on function pointers
    - Not type-safe
  - Observer Pattern (Listener)
    - Based on interface classes
    - Needs listener registration
    - Many interface classes
- Qt uses
  - Signals and slots for high-level (semantic) callbacks
  - Virtual methods for low-level (syntactic) events.

# Connecting Signals to Slots

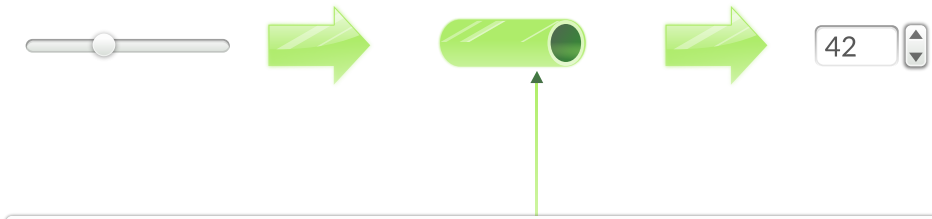
# Connecting Signals to Slots

# Connecting Signals to Slots

# Connecting Signals to Slots



# Connecting Signals to Slots




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

# Connecting Signals to Slots

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

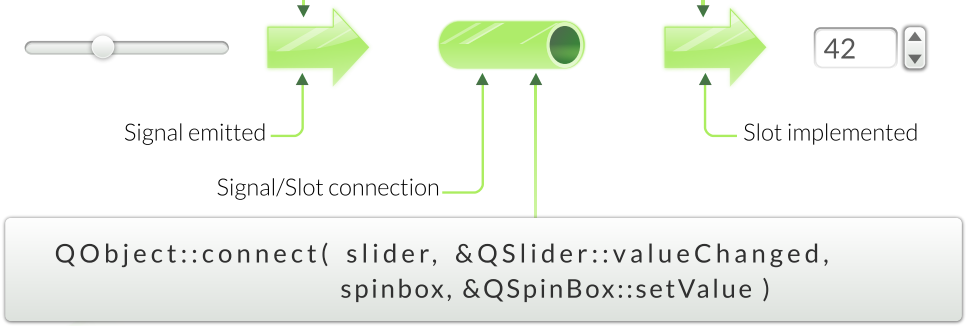
# Connecting Signals to Slots

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

# Connecting Signals to Slots

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

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




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

Demo object-communication/ex-connect

- Qt 4 style:

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

- Using function pointers:

```
connect( slider, &QSlider::valueChanged,
         spinbox, &QSpinBox::setValue );
```

- Using non-member function:

```
static void printValue(int value) {...}
connect( slider, &QSignal::valueChanged, &printValue );
```

- Using C++11 lambda functions:

```
connect( slider, &QSlider::valueChanged,
         [=] (int value) {...} );
```

# connect - function pointers

- Qt 5 components

```
connect( 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

X connecting to overloaded slots is hard

Demo object-communication/ex-connect-function-pointers

- Qt 4 ported code:

```
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

Demo object-communication/ex-connect

- File: **myclass.h**

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

- File: **myclass.cpp**

```
void MyClass::setValue(int value) {
  // slot implementation
}
```

Demo object-communication/ex-stop-watch

- Using non-member functions:

```
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

Demo object-communication/ex-connect-non-member

- Using C++11 lambda functions:

```
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

Demo object-communication/ex-connect-lambda

- File: **myclass.h**

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

- File: **myclass.cpp**

```
// No implementation for a signal
```

- Sending a signal

```
emit valueChanged(value);
```

Demo object-communication/ex-quotebutton

# `Q_OBJECT` - flag for MOC

- **`Q_OBJECT`**
  - 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

| Signal(s) | Connect to | Slot(s) |
|---|---|---|
| one | √ | many |
| many | √ | one |
| one | √ | another signal |

- Signal to Signal connection

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

- **Not** allowed to name parameters

```
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

| Signal | | Slot |
|---|---|---|
| rangeChanged(int,int) | √ | setRange(int,int) |
| | √ | setValue(int) |
| | √ | update() |
| valueChanged(int) | √ | setValue(int) |
| | √ | update() |
| | X | setRange(int,int) |
| | √ | setValue(float)† |
| textChanged(QString) | X | setValue(int) |

† 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 object-communication/lab-selectcolor

# 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.

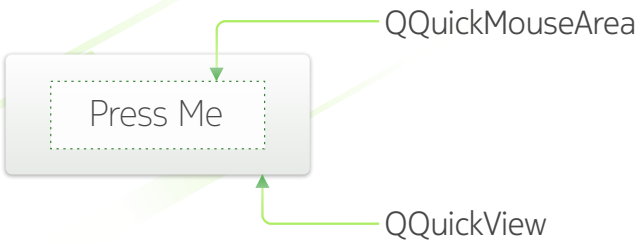
- Signals & Slots
- Event Handling

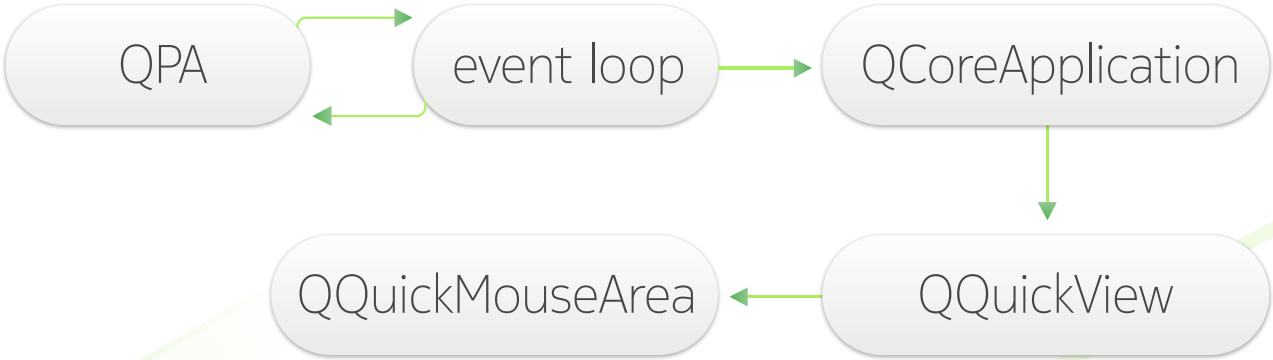
## Qt is an event-driven UI toolkit

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

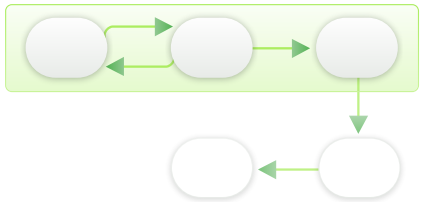
1. **Generate Events**
   - by input devices: keyboard, mouse, etc.
   - by Qt itself (e.g. timers)
2. **Queue Events**
   - by event loop
3. **Dispatch Events**
   - by `QApplication` to receiver: `QObject`
     - *Key events sent to widget with focus*
     - *Mouse events sent to widget under cursor*
4. **Handle Events**
   - by `QObject` event handler methods

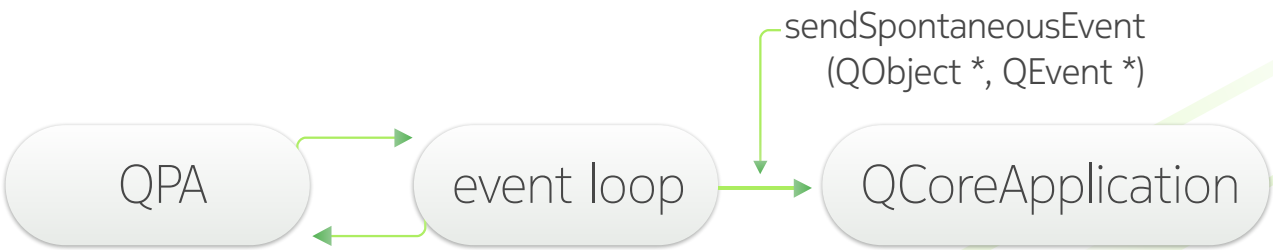
# Event Processing



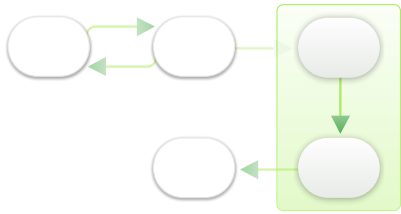



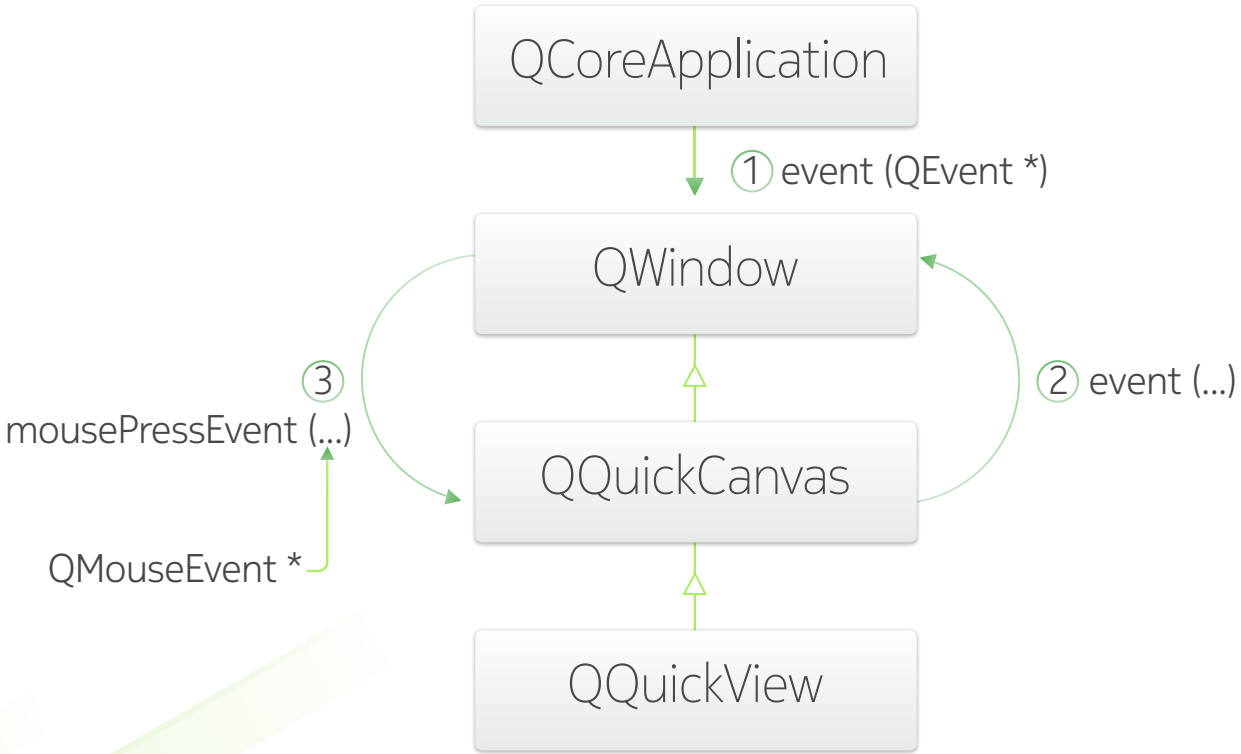
# Event Processing



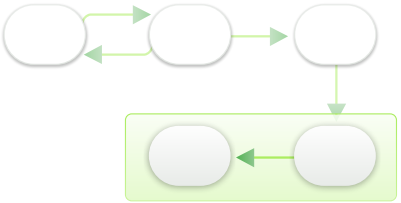


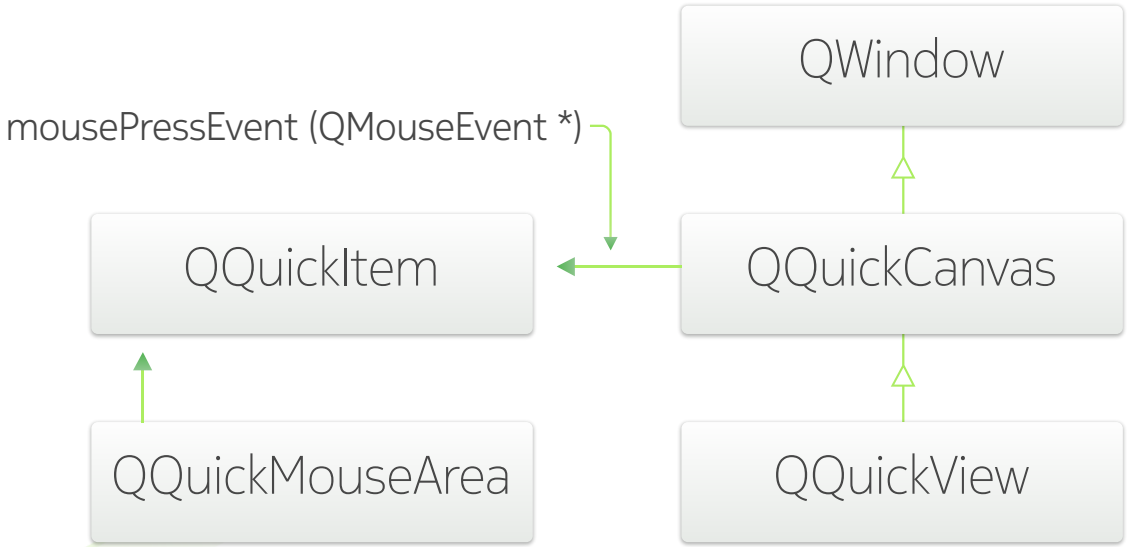
# Event Processing





# Event Processing






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

- `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

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

Demo object-communication/ex-closeevent

# Multi threaded object communication

- Signal/slots between threads
- Posting events using

```
QCoreApplication::postEvent(QObject* receiver,
                            QEvent* event)
```

- 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?