Qt Developer Days 2013

Qt Signals and Slots

Olivier Goffart

October 2013

About Me

About Me

- QStyleSheetStyle
- Itemviews
- Animation Framework
- QtScript (porting to JSC and V8)
- QObject, moc
- QML Debugger
- Modularisation
- ...

About Me

Offering Qt help and services: Visit `http://woboq.com`

C++ Code browser: `http://code.woboq.org`

Outline

Qt Developer Days 2013

woboq

Outline

Qt 1.41

Qt Developer Days 2013

ON-LINE REFERENCE DOCUMENTATION

Reference Documentation

The Qt API:

- Alphabetical Class List
- Annotated Class List
- Structure Overview
- Inheritance Hierarchy
- Alphabetical Function Index (long)
- Header File Index
- Widget Screenshots

Using Qt:

- Tutorial
- Example Programs
- Introduction to Signals and Slots
- Using the Meta Object Compiler (moc)
- Debugging Techniques

Qt Extensions:

- OpenGL 3D Graphics Support
- Netscape/Explorer Plugin Support
- Image File Formats Support
- Xt/Motif Legacy Code Support

Further Information:

- Troll Tech Contact Information
- Qt Mailing Lists
- Credits
- Developers' Homepage [external]

About Qt

Qt™ is a multi-platform C++ GUI toolkit. It is a product of Troll Tech. It is supported on all major variants of Microsoft Windows and Unix/X Windows.

Qt Professional Edition is provided for commercial software development. It is provided with upgrades and technical support. For the latest prices, please see the Troll Tech web site, Pricing and Availability page, or contact sales@troll.no.

Qt Free Edition is the Unix/X11 version of Qt available for development of *free software* only. It is provided free of charge under the Qt Free Edition License. The latest version is available for download.

 Trademarks Qt version 1.41

woboq

A Small Example

A minimal C++ class declaration might read:

```
class Foo
{
public:
    Foo();
    int  value() const { return val; }
    void setValue( int );
private:
    int  val;
};
```

A small Qt class might read:

```
class Foo : public QObject
{
    Q_OBJECT
public:
    Foo();
    int  value() const { return val; }
public slots:
    void setValue( int );
signals:
    void valueChanged( int );
private:
    int  val;
};
```

Slots are implemented by the application programmer (that's you). Here is a possible implementation of Foo::setValue():

```
void Foo::setValue( int v )
{
    if ( v != val ) {
        val = v;
        emit valueChanged(v);
    }
}
```

The line `emit valueChanged(v)` emits the signal `valueChanged` from the object. As you can see, you emit a signal by using `emit signal(arguments)`.

Here is one way to connect two of these objects together:

```
Foo a, b;
connect(&a, SIGNAL(valueChanged(int)), &b, SLOT(setValue(int)));
b.setValue( 11 );
a.setValue( 79 );
b.value();          // this would now be 79, why?
```

Qt 1.41 qobjectdefs.h

Qt Developer Days 2013

```
// The following macros are our "extensions" to C++
// They are used, strictly speaking, only by the moc.

#define slots                   // slots:   in class
#define signals protected       // signals: in class
#define emit                    // emit signal

/* tmake ignore Q_OBJECT */
#define Q_OBJECT                                                \
public:                                                         \
    QMetaObject *metaObject() const { return metaObj; }        \
    const char  *className()  const;                            \
protected:                                                      \
    void         initMetaObject();                              \
private:                                                        \
    static QMetaObject *metaObj;

/* tmake ignore Q_OBJECT */
#define Q_OBJECT_FAKE Q_OBJECT
                                // macro for naming members
#if defined(_OLD_CPP_)
#define METHOD(a)       "0""a"
#define SLOT(a)         "1""a"
#define SIGNAL(a)       "2""a"
#else
#define METHOD(a)       "0"#a
#define SLOT(a)         "1"#a
#define SIGNAL(a)       "2"#a
#endif

#define METHOD_CODE     0       // member type codes
#define SLOT_CODE       1
#define SIGNAL_CODE     2
```

woboq

Qt 2, Qt 3

- `Q_PROPERTY`
- No major changes in signals and slot

Qt 4

- Thread support
- `QueuedConnection`
- Meta type registration
- Several major internal changes
- Added file and line number information in debug mode
- But still no changes in the syntax

How Does it Work?

Qt Developer Days 2013

```
1 bool connect(const QObject *sender,
2               const char *signal,
3               const QObject *receiver,
4               const char *member);
```

woboq

How Does it Work?

- Compare the signature string to see if the arguments match
- Use the information provided my the `moc` to find the index of the signal and of the slot
- Keep in an internal map which signal is connected to what slots
- When emitting a signal, `QMetaObject::activate` is called.
- It calls `qt_metacall` (generated by `moc`) with the slot index which call the actual slot

Problems


```
1    connect(button, SIGNAL(clicked()),
2            this, SLOT(slotCliked()));
```

Problems


```
1    connect(button, SIGNAL(clicked()),
2            this, SLOT(slotCliked()));

3    connect(socket, SIGNAL(infoReceived(const Info &)),
4            this, SLOT(slotInfoReceived(const MyFramework::Info &)))
```

Problems


```
1    connect(button, SIGNAL(clicked()),
2            this, SLOT(slotCliked()));

3   connect(socket, SIGNAL(infoReceived(const Info &)),
4           this, SLOT(slotInfoReceived(const MyFramework::Info &)))

6    connect(button3, SIGNAL(clicked()),
7            this, SLOT(buttonClicked(3)));
```

Problems


```
1   connect(button, SIGNAL(clicked()),
2           this, SLOT(slotCliked()));

3   connect(socket, SIGNAL(infoReceived(const Info &)),
4           this, SLOT(slotInfoReceived(const MyFramework::Info &)))

6   connect(button3, SIGNAL(clicked()),
7           this, SLOT(buttonClicked(3)));

9   connect(comboBox, SIGNAL(valueChanged(int)),
10          settings, SLOT(updateValue(QVariant)));
```

Problems


```
1    connect(button, SIGNAL(clicked()),
2            this, SLOT(slotCliked()));

3   connect(socket, SIGNAL(infoReceived(const Info &)),
4           this, SLOT(slotInfoReceived(const MyFramework::Info &)))

6    connect(button3, SIGNAL(clicked()),
7            this, SLOT(buttonClicked(3)));

9    connect(comboBox, SIGNAL(valueChanged(int)),
10           settings, SLOT(updateValue(QVariant)));

12   connect(model, SIGNAL(modelReset()),
13           this, SLOT(oneLineSlot()));
```

Qt 5 syntax


```
1  connect(action, SIGNAL(selected(QString)),
2          receiver, SLOT(actionSelected(QString)));
3
4  connect(action, &QAction::selected,
5          receiver, &Receiver::actionSelected);
6
7  connect(action, &QAction::selected,
8          [](const QString &act) {
9      qDebug() <<  "Action selected:" << act;
10 });
```

woboq

Outline

Pointer to member


```
1  struct Point { int x;  int y; };
2
3  int Point::*coordinate = 0;
4
5  if (orientation == Qt::Horizontal)
6      coordinate = &Point::x;
7  else if (orientation == Qt::Vertical)
8      coordinate = &Point::y;
9
10 Point p = /* ... */
11 Point *pp = /* ... */
12
13 if (coordinate)
14     pp->*coordinate = p.*coordinate;
```

Pointer to member function


```
1  struct Point { int x() const; int y() const;
2                 void setX(int); void setY(int); };
3
4  int (Point::*getter)() const = 0;
5  void (Point::*setter)(int) = 0;
6
7  if (orientation == Qt::Horizontal) {
8      getter = &Point::x;
9      setter = &Point::setX;
10 }
11
12 Point p = /* ... */
13 Point *pp = /* ... */
14
15 if (getter && setter) {
16     int c = (p.*getter)()
17     (pp->*setter)(c);
18 }
```

woboq

Fun facts


```
1    int Point::*coordinate = 0;
2    int (Point::*getter)() = 0;
```

Fun facts


```
1     int Point::*coordinate = 0;
2     int (Point::*getter)() = 0;

3   qDebug() << sizeof(coordinate) << sizeof(getter);
```

Fun facts


```
1    int Point::*coordinate = 0;
2    int (Point::*getter)() = 0;

3  qDebug() << sizeof(coordinate) << sizeof(getter);
```

```
8   16
```

Fun facts


```
1    int Point::*coordinate = 0;
2    int (Point::*getter)() = 0;

3   qDebug() << sizeof(coordinate) << sizeof(getter);
```

8 16

```
4   qDebug() << *reinterpret_cast<int*>(&coordinate);
```

woboq

Fun facts


```
1    int Point::*coordinate = 0;
2    int (Point::*getter)() = 0;
3   qDebug() << sizeof(coordinate) << sizeof(getter);
```

8 16

```
4   qDebug() << *reinterpret_cast<int*>(&coordinate);
```

-1

Not So Fun Facts


```
1  struct Struct {
2    int foo(int);
3    int bar(int);
4    int bar(double);
5  };

6    int (Struct::*barP1)(int) = &Struct::bar;
```

Not So Fun Facts

Qt Developer Days 2013

```
1  struct Struct {
2    int foo(int);
3    int bar(int);
4    int bar(double);
5  };

6    int (Struct::*barP1)(int) = &Struct::bar;

7    auto fooP = &Struct::foo;
```

woboq

Not So Fun Facts


```
1  struct Struct {
2    int foo(int);
3    int bar(int);
4    int bar(double);
5  };

6    int (Struct::*barP1)(int) = &Struct::bar;

7    auto fooP = &Struct::foo;

8    // decltype(fooP):   int (Struct::*)(int)
```

woboq

Not So Fun Facts


```
1  struct Struct {
2    int foo(int);
3    int bar(int);
4    int bar(double);
5  };

6    int (Struct::*barP1)(int) = &Struct::bar;

7    auto fooP = &Struct::foo;

8    // decltype(fooP):   int (Struct::*)(int)

10   auto barP2 = &Struct::bar;
```

Not So Fun Facts


```
1  struct Struct {
2    int foo(int);
3    int bar(int);
4    int bar(double);
5  };

6    int (Struct::*barP1)(int) = &Struct::bar;

7    auto fooP = &Struct::foo;

8    // decltype(fooP):   int (Struct::*)(int)

10   auto barP2 = &Struct::bar;
```

```
error: variable 'barP2' with type 'auto' has incompatible initializer
       of type '<overloaded function type>'

error: unable to deduce 'auto' from '& Struct::bar'
```

Outline

Lambda

Lambda


```
[foo] (int a) -> int { return a + foo; }
```

- Capture: Variables that you capture
- Parametter list: The perametters of the function
- Return type (optional)
- Function body

Lambda


```
[foo] (int a) -> int { return a + foo; }
```

```
struct {
    double foo;
    int operator()(int a)
    { return a + foo; }
}
```

Lambda capture


```
1  int a = 1, b = 2, c = 3;
2
3  // 'a' by value, 'b' by reference
4  auto f1 = [a, &b]() { b = a; };
5
6  // everything by reference
7  auto f2 = [&]() { b = a; };
8
9  // everything by value
10 auto f3 = [=]() { return a + c; };
11
12 // everything by value, 'b' by reference
13 auto f4 = [=,&b]() { b = a + c; };
```

Examples


```
1  connect(button3, &QBushButton::clicked,
2          [=] { this->buttonClicked(3); });
3
4  connect(model, &QAbstractItemModel::modelReset,
5          [] { qDebug() << "model has been reset"; });
```

Outline

New connect Overloads

1. `QObject::connect(const QObject *sender, const char *signal, const QObject *receiver, const char *slot, Qt::ConnectionType type)`
2. `QObject::connect(const QObject *sender, PointerToMemberFunction signal, const QObject *receiver, PointerToMemberFunction slot, Qt::ConnectionType type)`
3. `QObject::connect(const QObject *sender, PointerToMemberFunction signal, Functor method)`
4. `QObject::connect(const QObject *sender, PointerToMemberFunction signal, const QObject *context, Functor method, Qt::ConnectionType type)` (since Qt 5.2)

woboq

New connect Overloads

1. `QObject::connect(const QObject *sender, const char *signal, const QObject *receiver, const char *slot, Qt::ConnectionType type)`
2. `QObject::connect(const QObject *sender, PointerToMemberFunction signal, const QObject *receiver, PointerToMemberFunction slot, Qt::ConnectionType type)`
3. `QObject::connect(const QObject *sender, PointerToMemberFunction signal, Functor method)`
4. `QObject::connect(const QObject *sender, PointerToMemberFunction signal, const QObject *context, Functor method, Qt::ConnectionType type)` (since Qt 5.2)

woboq

Context

- There is no "receiver" when connecting to a lambda.
- Receiver can be used for:
  1. Thread affinity (`QueuedConnection`)
  2. Automatic disconnection when the receiver is destroyed
  3. `sender()`
- In Qt 5.2 you can use a context with your lambda function

```
1 connect(button3, &Button::clicked,
2         bar, [=]{ bar->buttonClicked(3); });
```

Remember QSignalMapper?


```
1  for (int i = 0; i < texts.size(); ++i) {
2      QPushButton *button = new QPushButton(texts[i],
3                                             this);
4
5      // A C++11 lambda
6      connect(button, &QPushButton::clicked,
7              [=]{ this->select(texts[i]) });
8
9      // OR, without C++11, using tr1/boost bind
10     connect(button, &QPushButton::clicked,
11             bind(&MyWidget::select, this, texts[i]));
12 }
```

Qt 5 syntax

Qt Developer Days 2013

Design Goals

- Detect as many error as possible at compile time
- Be easy and intuitive
- Do not require users to understand templates or function pointers
- Do not require C++11

woboq

Qt 5 syntax

Qt Developer Days 2013

History

- Research started in august 2009 ($\sim$ Qt 4.6)
- First trial:

```
1 connect(QSignal(s, &SenderObject::signal1),
2           QSlot(r1, &ReceiverObject::slot1));
```

woboq

Protected Signals

In Qt4 (and before)

```
1  #define signals protected
```

```
37      connect(MyObject, &QObject::destroyed, //...
```

```
main.cc:37:22: error: 'destroyed' is a
protected member of 'QObject'
```

Protected Signals

Can we change?

```
1 #define signals public
```

Protected Signals

Qt Developer Days 2013

Can we change?

```
1 #define signals public
```

Two problems

- Binary compatibility
- Everybody can emit a signal from another object

woboq

Typing the type name

Is there a way to avoid typing the type of the object?

```
1       connect(QSIGNAL(button, clicked),
2               QSLOT(receiver, buttonClicked));
```

Typing the type name

Is there a way to avoid typing the type of the object?

```
1       connect(QSIGNAL(button, clicked),
2               QSLOT(receiver, buttonClicked));
```

```
10  // C++11 Only
11  #define QSIGNAL(OBJ, FUNC) OBJ,  [&](){ \
12    typedef std::remove_reference<decltype(*(OBJ))>::type Type; \
13    return &Type::FUNC; }()
```

Overloads


```
1
2     class Obj : public QObject {
3       Q_OBJECT
4       signals:
5         void valueChanged(int);
6         void valueChanged(const QString &);
7     };
8
9     QObject::connect(obj, &Obj::valueChanged, []{});
```

Overloads


```
1
2     class Obj : public QObject {
3       Q_OBJECT
4       signals:
5         void valueChanged(int);
6         void valueChanged(const QString &);
7     };
8
9     QObject::connect(obj, &Obj::valueChanged, []{});
```

```
error: no matching function for call to ‘QObject::connect(Obj* const,
       <unresolved overloaded function type>, __lambda0)’
```

woboq

Overloads


```
1
2     class Obj : public QObject {
3       Q_OBJECT
4       signals:
5         void valueChanged(int);
6         void valueChanged(const QString &);
7     };
8
9     QObject::connect(obj, &Obj::valueChanged, []{});
```

```
error: no matching function for call to ‘QObject::connect(Obj* const,
       <unresolved overloaded function type>, __lambda0)’
```

```
1   QObject::connect(obj,
2       static_cast<void(Obj::*)(int)>(&Obj::valueChanged),
3       []{});
```

Overloads

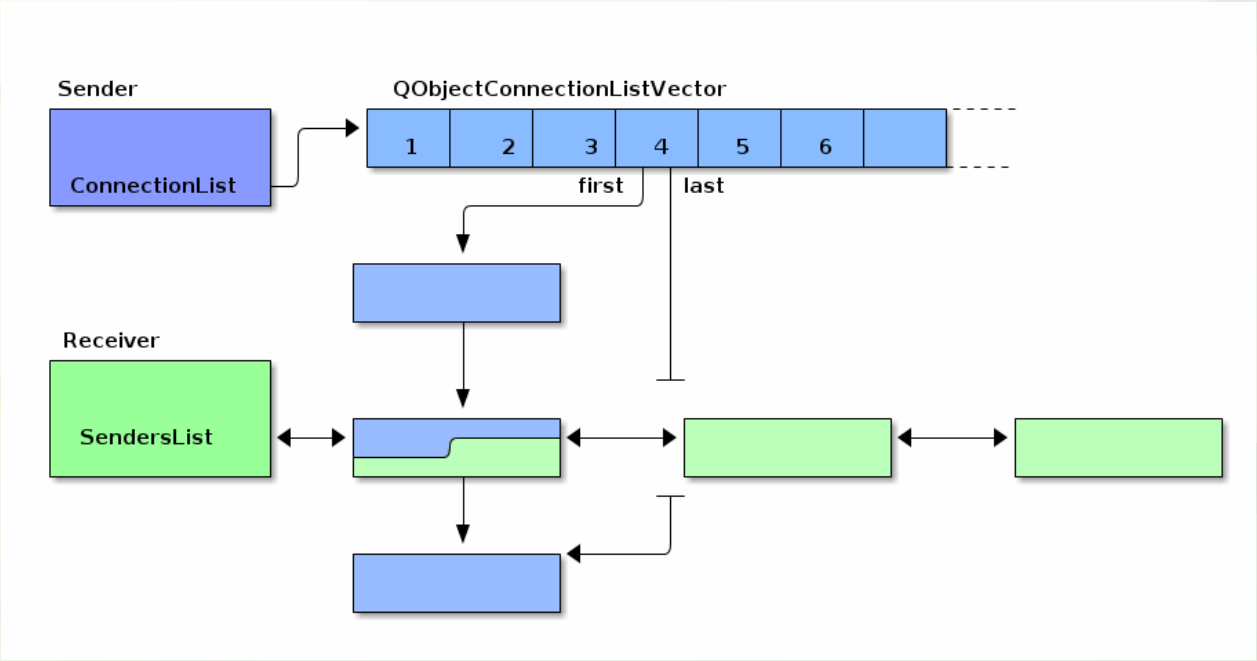
Avoid overloading signals!

Outline

Outline

Outline

Connections



Outline

Outline

New Syntax

Summary

- Compile time checks
- Not problems in arguments with namespaces or typedef
- Automatic type converstions

Summary

- Compile time checks
- Not problems in arguments with namespaces or typedef
- Automatic type converstions

With C++11 you benefit from

- No 6 arguments limit
- Better error messages `static_assert`
- Lambda functions

The END

Qt Developer Days 2013

Questions

olivier@woboq.com

woboq

visit http://woboq.com

woboq