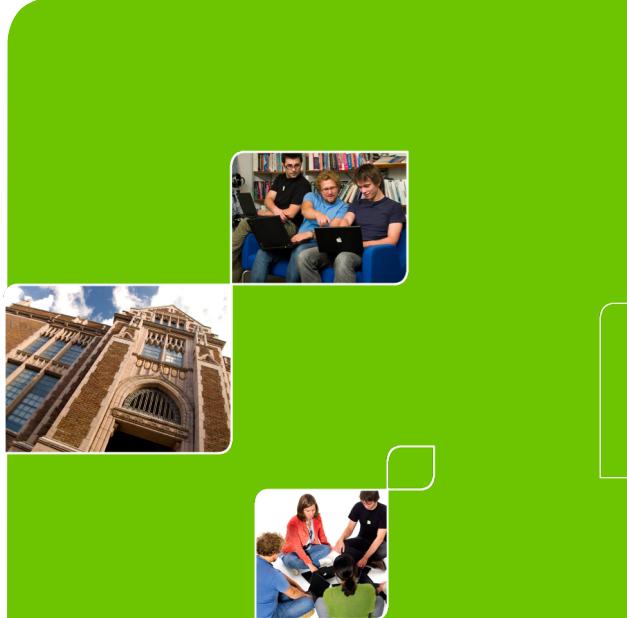




Qt dans l'Enseignement

Le modèle objet & Le concept de signal / “slot”



digia



Traduction française et Adaptation des supports pour l'enseignement

© 2012 Digia Plc.

The enclosed Qt Materials are provided under the Creative Commons Attribution-Share Alike 2.5 License Agreement.



The full license text is available here:
<http://creativecommons.org/licenses/by-sa/2.5/legalcode>.

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



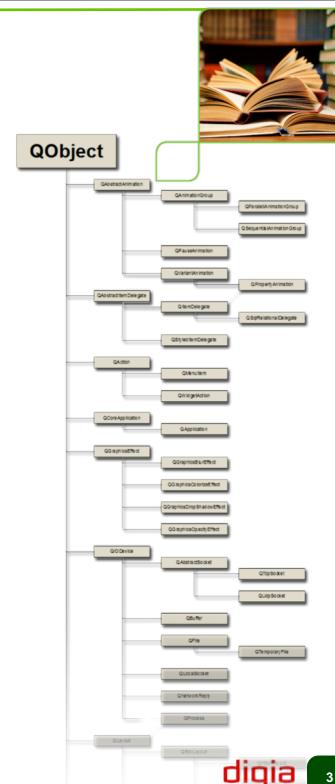
<http://www.qt.io/training-materials/>
<http://www.qt.io/qt-essentials-widget-edition/>

digia 2

The Qt logo consists of the letters "Qt" in a white, sans-serif font, enclosed within a thick, rounded green square.

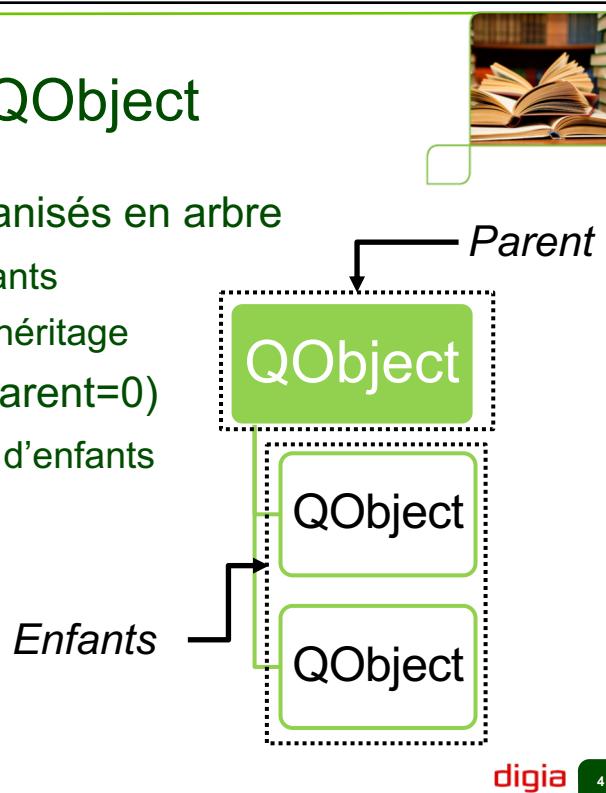
La classe QObject

- QObject = classe de base
 - De presque toutes classes Qt
 - Son rôle = gestion
 - événements
 - signals et slots
 - propriétés
 - Mémoire
 - Pas de représentation visuelle



La classe QObject

- Les objets sont organisés en arbre
 - Relation parent/enfants
 - Autre relation que l'héritage
 - QObject(Qobject *parent=0)
 - Parent gère sa liste d'enfants





La classe QObject

- Exemples de classes n'héritant pas de QObject
 - Classes qui doivent être allégées
 - Exemple : primitives graphiques
 - Les conteneurs de données
 - QString, QList, QChar, etc.
 - Classes qui ont besoin d'être copiées
 - Les objets QObjects ne peuvent pas être copiés

digia 5



La classe QObject

“Les instances QObject sont uniques !”

- Caractéristiques des instances QObject
 - possèdent un nom (QObject::objectName)
 - placées dans une hiérarchie d'instances Qobject
 - peuvent avoir des connexions avec d'autres instances

“ Est-ce que cela a un sens de copier un widget à l'exécution? ”

digia 6



Meta données



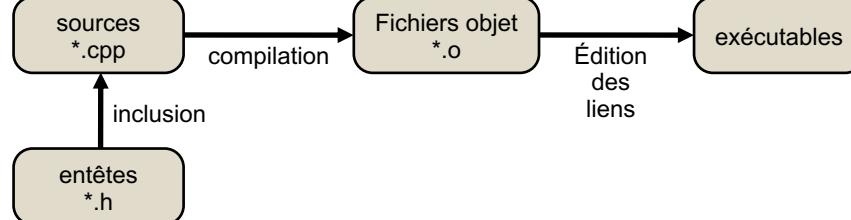
- Qt implémente l'introspection en C++
- Chaque QObject possède un meta objet
- Le meta objet connaît
 - le nom de la classe (QObject::className)
 - de qui il hérite (QObject::inherits)
 - ses propriétés
 - les signaux et les slots
 - des informations générales (QObject::classInfo)

digia 7



Meta données

- Définies à la compilation par MOC
 - Meta Object Compiler
- Processus de construction classique en C++**



digia 8

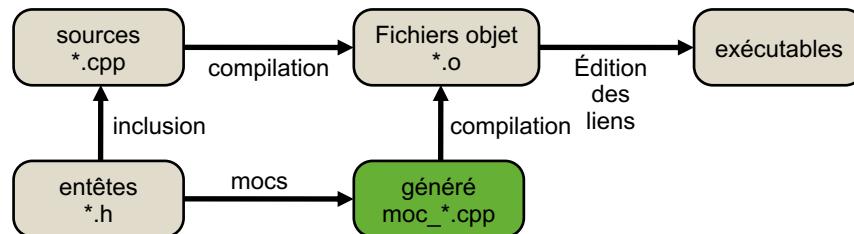


Meta données

- Définies à la compilation par MOC

- Meta Object Compiler

Processus de construction C++ avec Qt



- Le MOC récupère les données à partir des entêtes

digia 9



Meta données

- Ce qu'il faut ajouter dans les classes pour MOC

Macro en premier

```

class MyClass : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("author", "John Doe")

public:
    MyClass(const Foo &foo, QObject *parent=0);

    Foo foo() const;

public slots:
    void setFoo( const Foo &foo );

signals:
    void fooChanged( Foo );
private:
    Foo m_foo;
};
  
```

Héritage de QObject
(ou dérivée)

Informations générales sur la classe

Mots clés Qt

digia 10



Introspection



- Connaissance sur les classes à l'exécution
 - Intégration plus facile avec
 - langages de script
 - Autres tels que PyQt, Ruby, JavaScript, etc.
- Exemple 1 Le méta-objet connaît son héritage

```
if (object->inherits("QAbstractItemView"))
{
    QAbstractItemView *view = static_cast<QAbstractItemView*>(object);
    view->...
```

Transtypage dynamique sans utiliser RTTI

digia 11



Introspection



- Exemple 2

```
#include <QMenum>
```

Fichier entête

```
Class MyClass : Qobject
{
    Q_OBJECT()
    Q_ENUMS(CapitalsEnum)
public:
    enum CapitalsEnum { Oslo, Helsinki, Stockholm, Copenhagen };
...};
```

pour que le méta-objet
connaisse l'énumération

Position d'une énumération dans le méta-objet

```
int index = obj->metaObject()->indexOfEnumerator("CapitalsEnum");
QString text = obj->metaObject()->enumerator(index).key(obj->capital());
```

Accès à l'énumération dans le méta-objet

Conversion en chaîne d'une valeur du type énuméré

12



Propriétés



- QObject a des propriétés avec accesseurs

`Q_PROPERTY(<typepropriété nom> READ <consultation> WRITE <modification>)`

- Consultation : méthode const sans paramètre
 - Nom sans préfixe “get”, préfixe “is” si booléen
- Modification : méthode void avec un seul paramètre
 - Nom préfixé par “set”

- Exemple

```
class QLabel : public QFrame
{
    Q_OBJECT
    Q_PROPERTY(QString text READ text WRITE
    setText)
public:
    QString text() const;
public slots:
    void setText(const QString &);
```

digia

13



Propriétés

- Pourquoi une méthode de modification?

- Pour vérifier la validité de la nouvelle valeur

```
void setMin( int newMin )
{
    if( newMin > m_max )
    {
        qWarning("Ignoring setMin(%d) as min > max.", newMin);
        return;
    }
    ...
}
```

- Pour réagir aux modifications

```
void setMin( int newMin )
{
    ...
    m_min = newMin;
    updateMinimum();
}
```

digia

14



Propriétés

- Pourquoi une méthode de consultation?
- Séparer stockage de consultation ⇒ lecture indirecte

```
QSize size() const
{
    return m_size;
}

int width() const
{
    return m_size.width();
}
```

digia

15



Propriétés

- Paramétrage d'une propriété

```
Q_PROPERTY(type name
          READ getFunction
          [WRITE setFunction]
          [RESET resetFunction]
          [NOTIFY notifySignal]
          [DESIGNABLE bool]
          [SCRIPTABLE bool]
          [STORED bool]
          [USER bool]
          [CONSTANT]
          [FINAL] )
```

True par défaut

False par défaut

digia

16



Propriétés : utilisation

- Accès direct

```
QString text = label->text();
label->setText("Hello World!");
```

- Avec les métainformation et propriétés système

```
QString text = object->property("text").toString();
object->setProperty("text", "Hello World");
```

- Informations sur les propriétés à l'exécution

```
int QMetaObject::propertyCount();
QMetaProperty QMetaObject::property(i);

QMetaProperty::name/isConstant/isDesignable/read/write/...
```

digia

17



Propriétés dynamiques

- Ajout de propriétés à l'exécution

```
bool ret = object->setProperty(name, value);
```

True si name définie avec Q_PROPERTY
 False sinon + ajoutée dynamiquement

- Liste des propriétés dynamiques

```
QList<QByteArray> QObject::dynamicPropertyNames() const
```

- Usage : marquer un objet selon un contexte

digia

18



Propriétés personnalisées



- Suivre le modèle Qt
 - Stocker la propriété dans une donnée privée
 - Définir une méthode de consultation
 - const
 - Sans paramètre
 - Définir une méthode de modification
 - void
 - 1 seul paramètre
 - Initialiser la propriété
 - Utiliser la macro Q_PROPERTY

digia 19



Propriétés personnalisées



- Exemple

```
class AngleObject : public QObject
{
    Q_OBJECT
    Q_PROPERTY(qreal angle READ angle WRITE setAngle)

public:
    AngleObject(qreal angle, QObject *parent = 0);

    qreal angle() const;           ← Consultation
    void setAngle(qreal);          ← Modification

private:
    qreal m_angle;                ← Donnée privée
};
```

Valeur initiale

Consultation

Modification

Donnée privée

digia 20



Propriétés personnalisées

```
AngleObject::AngleObject(qreal angle, QObject *parent) :
    QObject(parent), m_angle(angle)
{
}

qreal AngleObject::angle() const
{
    return m_angle;
}

void AngleObject::setAngle(qreal angle)
{
    m_angle = angle;
    doSomething();
}
```

Valeur initiale

digia 21



Propriétés personnalisées

- Cas où type = énumération \Rightarrow macro Q_ENUMS

```
class AngleObject : public QObject
{
    Q_OBJECT
    Q_ENUMS (AngleMode)
    Q_PROPERTY(AngleMode angleMode READ ...)

public:
    enum AngleMode {Radians, Degrees};
    ...
};
```

Macro indiquant à Qt
qu'AngleMode est un
type énuméré

Énumération définie
classiquement

digia 22



Gestion de la mémoire

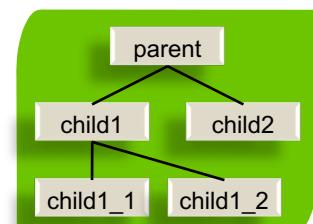


- QObject a un parent et des enfants
- Parent détruit ⇒ il détruit ses enfants

```
QObject *parent = new QObject();
QObject *child1 = new QObject(parent);
QObject *child2 = new QObject(parent);
QObject *child1_1 = new QObject(child1);
QObject *child1_2 = new QObject(child1);
```

`delete parent;`

parent détruit child1 et child2
child1 détruit child1_1 et child1_2



digia 23



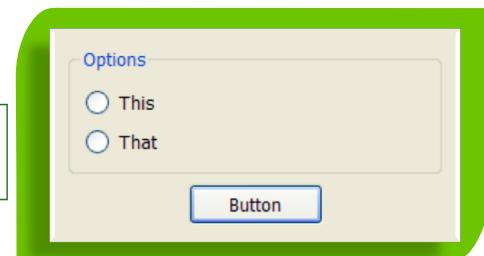
Gestion de la mémoire

- Utilisé dans les hiérarchies visuelles aussi

```
QDialog *parent = new QDialog();
QGroupBox *box = new QGroupBox(parent);
QPushButton *button = new QPushButton(parent);
QRadioButton *option1 = new QRadioButton(box);
QRadioButton *option2 = new QRadioButton(box);
```

`delete parent;`

parent détruit box et button
box détruit option1 et option2



digia 24



Gestion de la mémoire

- Enfants \Rightarrow alloués dans constructeur du parent

```
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{
    QGroupBox *box = new QGroupBox(this);
    QPushButton *button = new QPushButton(this);
    QRadioButton *option1 = new QRadioButton(box);
    QRadioButton *option2 = new QRadioButton(box);
    ...
}
```

- Parent défini sur la pile

```
void Widget::showDialog()
{
    Dialog dialog;

    if (dialog.exec() == QDialog::Accepted)
    {
        ...
    }
}
```

Dialog détruit automatiquement

digia 25



Rappel : tas \neq pile

- Allocation sur le tas \Rightarrow new et delete
- Mémoire sur le tas \Rightarrow libérée explicitement
 - Avec delete
 - But : éviter les fuites mémoire
- Durée de vie objet sur tas ?
 - Aussi longtemps que nécessaire

new
Construction



destruction

delete

digia 26

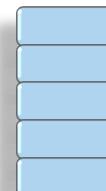


Rappel : tas ≠ pile

- Variables locales \Rightarrow sur la pile
- Variables sur pile \Rightarrow destruction automatique
 - Quand hors de portée

int a

Construction



Destruction

}

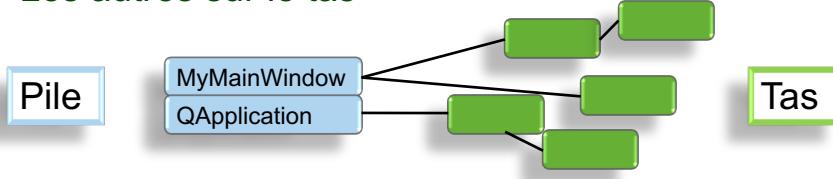
digia

27



Rappel : tas ≠ pile

- Gestion automatique complète de la mémoire
 - Que les parents de plus haut niveau sur la pile
 - Les autres sur le tas



```
int main(int argc, char **argv)
{
    QApplication a(argc, argv);
    MyMainWindow w;
    w.show();
    return a.exec();
}
```

```
MyMainWindow::MyMainWindow(... {
    new QLabel(this);
    new ...
})
```

digia

28

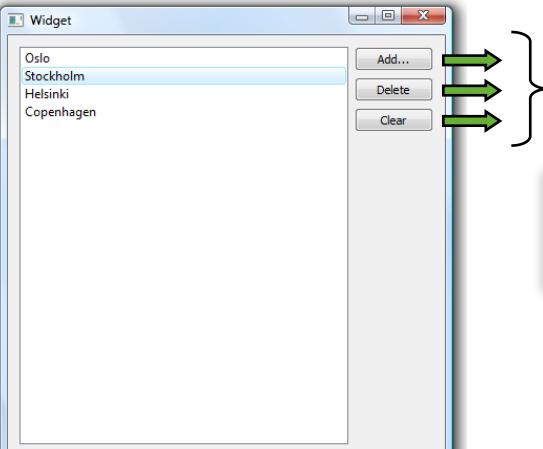
Qt Signaux / slots



- Association
 - dynamique ⇒ à l'exécution
 - simple ⇒ source et destinataire ne se connaissent pas
 - entre événement ⇒ clics, compteur...pas QEvent
 - et changement d'état ⇒ de taille, de texte, de valeur, ...
 - avec des réactions ⇒ du code qui agit
- Ce qui fait la particularité de Qt

digia 29

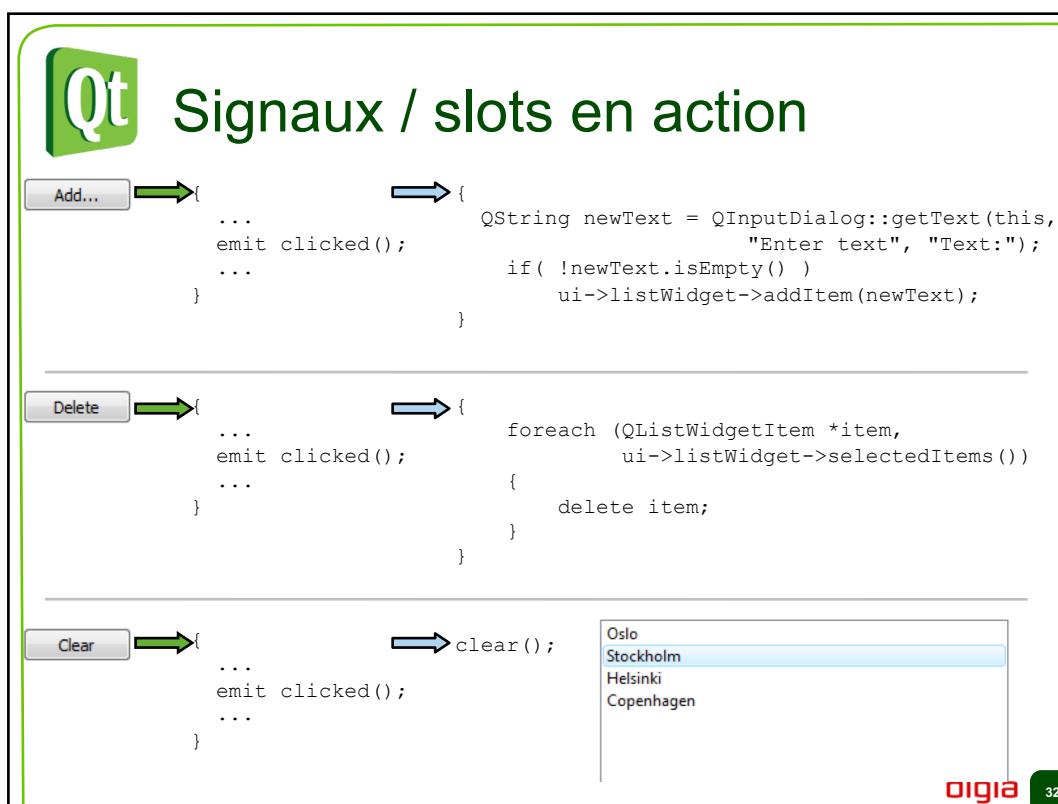
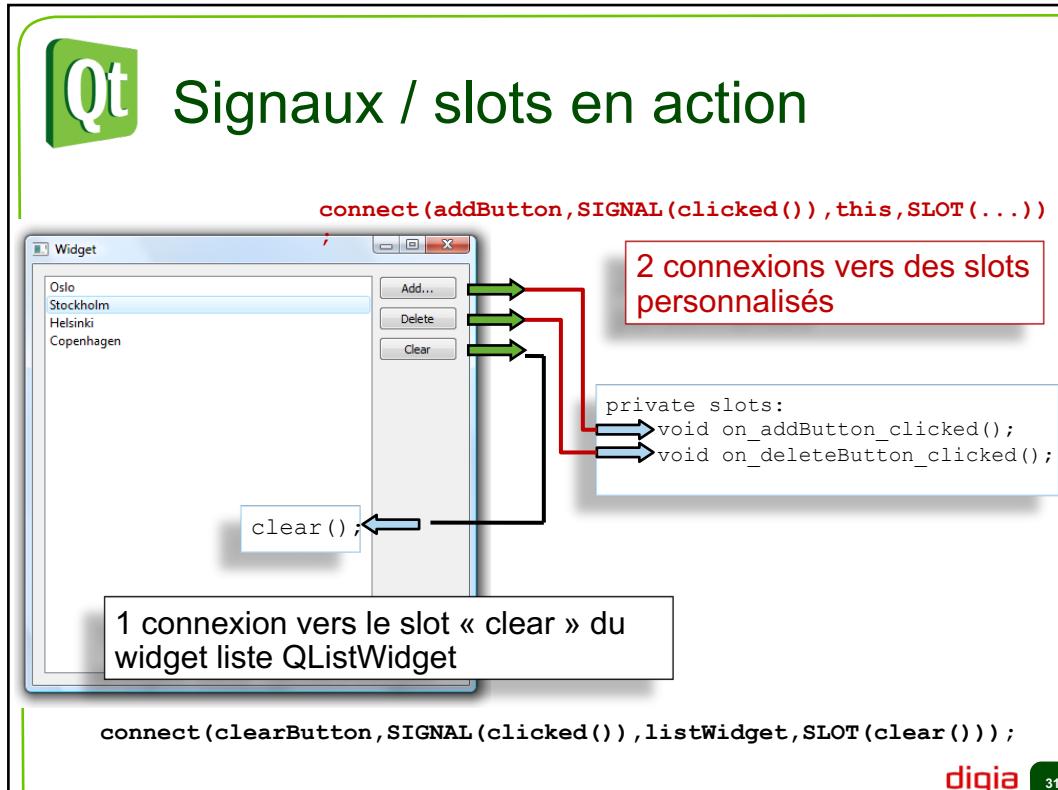
Qt Signaux / slots en action



emit clicked();

Signal émis sans se soucier
-s'il y a des connexions
-quelles sont-elles

digia 30





Signaux / slots vs Callbacks



- Callback = pointeur de fonction
 - Appelé quand événement se produit
 - Toute fonction peut être affectée à un callback
 - Pas de sécurité au niveau du type
 - Fonctionne toujours par appel direct
- Signaux/slots ⇒ plus dynamique
 - Mécanisme générique
 - Plus simple de connecter 2 classes existantes
 - Moins de connaissance partagée entre les 2 classes

digia 33



Qu'est-ce qu'un slot?

- Fonction classique déclarée dans section “slots”

```
public slots:
    void aPublicSlot();
protected slots:
    void aProtectedSlot();
private slots:
    void aPrivateSlot();
```

Restriction prise en compte que si utilisé comme une fonction

- Caractéristique : connecté à 1 ou n signaux

```
connect(src, SIGNAL(sig()), dest, SLOT(slt()));
```

- peut retourner des valeurs
- mais pas à travers les connexions !

digia 34



Qu'est-ce qu'un signal?

- Fonction

- void
- définie dans la section signals \Rightarrow protégé
- sans implémentation car définie par moc

```
signals:  
    void aSignal();
```

- Caractéristique

- connecté à 1 ou n slots ou à un autre signal

- Emission du signal

- Slots connectés sont exécutés
- Leur ordre d'appel est arbitraire

```
emit aSignal();
```

- Connexion possible à travers thread ou socket

35



La connexion signal/slot



- Style Qt4

```
QObject::connect(src, SIGNAL(signature), dest, SLOT(signature));
```

QObject*

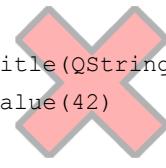
=

- Signature = nom de fonction + types d'arguments

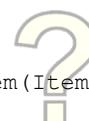
- Pas de noms de variable
- Pas de valeurs
- Éviter les types personnalisés
 - Limite la réutilisation

clicked()
toggled(bool)
setText(QString)
textChanged(QString)
rangeChanged(int,int)

setTitle(QString text)
setValue(42)



setItem(ItemClass)

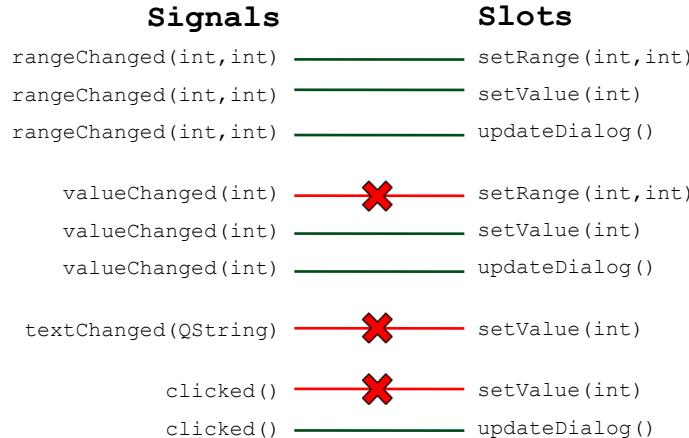


36



La connexion signal/slot

- Correspondance des signatures signal/slot
 - Qt peut ignorer des arguments
 - Mais ne peut pas créer des valeurs à partir de rien



digia 37



La connexion signal/slot



- Style QT4
 - :(Besoin de déclarer slot dans section « slots » »
 - :(Besoin de la macro Q_OBJECT
 - :(Besoin d'exécuter le MOC
 - :(Erreurs à l'exécution seulement
 - :) Surcharger un slot est facile
 - :) Code QT4 toujours supporté

digia 38



La connexion signal/slot



- Connexion avec pointeur de fonction (depuis Qt 5)

```
QObject::connect(src, &fct, dest, &fct));
```

- Exemple

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

- ☺ Erreurs à la compilation générées
- ☺ Q_OBJECT n'a pas besoin de slots
- ☺ Pas de syntaxe spéciale pour les slots
- ☹ Connexion avec slots surchargés difficile

digia 39



La connexion signal/slot



- Connexion avec fonction non membre

- Exemple

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

- ☺ Erreurs à la compilation générées

- ☺ Q_OBJECT n'a pas besoin de slots

- ☺ Pas de syntaxe spéciale pour les slots

- ☺ Pas de MOC

- ☺ Fonction quelconque

digia 40



La connexion signal/slot



- Connexion avec expression lambda (en C++11)

- Exemple

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

- ☺ Erreurs à la compilation générées
- ☺ Q_OBJECT n'a pas besoin de slots
- ☺ Pas de syntaxe spéciale pour les slots
- ☺ Pas de MOC
- ☺ Pas besoin de fonction particulière

digia 41



Connexions automatiques

- Dans Designer entre l'interface et le code

```
on_object_name_signal_name ( signal_parameters ) ← Slot généré
on_addButton_clicked();
on_deleteButton_clicked();
on_listWidget_currentItemChanged(QListWidgetItem*, QListWidgetItem*)
```

- Connexions faites par `QMetaObject::connectSlotsByName`
 - Appelée dans fonction `setupUI` générée par Designer
 - Attention aux noms de slots ⇒ réutilisation
 - Comparer `on_widget_signal` à `updatePageMargins`

Le slot pourra plus clairement être utilisé par plusieurs signaux ou par appel direct ⇒ mieux vaut parfois la connexion manuelle !

digia 42



Synchronisation de widgets



- Connexion dans les deux sens

```
connect(dial1, SIGNAL(valueChanged(int)), dial2, SLOT(setValue(int)));
```



```
connect(dial2, SIGNAL(valueChanged(int)), dial1, SLOT(setValue(int)));
```

- Arrêt de la boucle infinie

- pas de signal émis si pas de nouvelle valeur

```
void QDial::setValue(int v)
{
    if(v==m_value)
        return;
    ...
}
```

Ce test devrait exister pour tout code qui émet un signal

digia 43



Signaux et slots propres



```
class AngleObject : public QObject
{
    Q_OBJECT
    Q_PROPERTY(qreal angle READ angle WRITE setAngle NOTIFY angleChanged)

public:
    AngleObject(qreal angle, QObject *parent = 0);
    qreal angle() const;

public slots:
    void setAngle(qreal);

signals:
    void angleChanged(qreal);

private:
    qreal m_angle;
};
```

Indispensable pour gérer signaux/slots

Ajout d'un signal de notification de changement de valeur de la propriété définie

Les modificateurs font des slots naturels

Signaux correspondant aux modificateurs

digia 44



Signaux et slots propres

- Implémentation du modifieur : détail

Mise à jour de la propriété, puis émission du signal

```
void AngleObject::setAngle(qreal angle)
{
    if(m_angle == angle)
        return;
    m_angle = angle;
    emit angleChanged(m_angle);
}
```

Protection contre les boucles infinies
Ne pas oublier!

- Emettre signal dans classe dérivée ⇒ ok
 - Car les signaux sont considérés protégés

digia

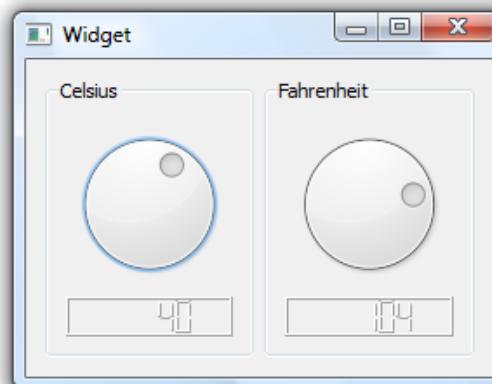
45



Signaux et slots : exemple



- Application : convertisseur ° F ⇔ ° C
 - classe TempConverter pour la conversion
 - Signaux émis quand température change



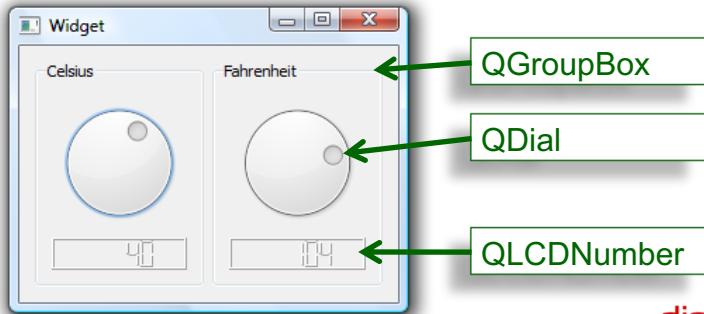
digia

46

Qt Signaux et slots : exemple



- La fenêtre QDialog contient les objets suivants
 - Une instance TempConverter
 - 2 QGroupBox, contenant chacun
 - Un cadran QDial
 - Un afficheur LCD QLCDNumber



digia 47

Qt Signaux et slots : exemple

```

class TempConverter : public QObject ← Hérite de QObject
{
    Q_OBJECT ← La macro Q_Object en premier
public:
    TempConverter(int tempCelsius, QObject *parent = 0);

    int tempCelsius() const;
    int tempFahrenheit() const;
}

public slots:
    void setTempCelsius(int);
    void setTempFahrenheit(int);
}

signals:
    void tempCelsiusChanged(int);
    void tempFahrenheitChanged(int); } Émis quand les
températures changent

private:
    int m_tempCelsius; ← Représentation interne de
la température en Celsius
};

```

digia 48



Signaux et slots : exemple

- Le slot `setTempCelsius`

```
void TempConverter::setTempCelsius(int tempCelsius)
{
    if(m_tempCelsius == tempCelsius) ← Test pour éviter boucle infinie
        return;
    m_tempCelsius = tempCelsius; ← Mise à jour de la température courante
    emit tempCelsiusChanged(m_tempCelsius);
    emit tempFahrenheitChanged(tempFahrenheit()); } } ] Emission des signaux pour indiquer le changement
```

- Le slot `setTempFahrenheit`

```
void TempConverter::setTempFahrenheit(int tempFahrenheit)
{
    int tempCelsius = (5.0/9.0)*(tempFahrenheit-32); ← Conversion en ° C
    setTempCelsius(tempCelsius); ← Mise à jour de la représentation interne }
```

digia

49



Signaux et slots : exemple

- Cadrans ⇒ interconnectés via `TempConverter`

```
connect(celsiusDial, SIGNAL(valueChanged(int)),
        tempConverter, SLOT(setTempCelsius(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)),
        celsiusDial, SLOT(setValue(int)));

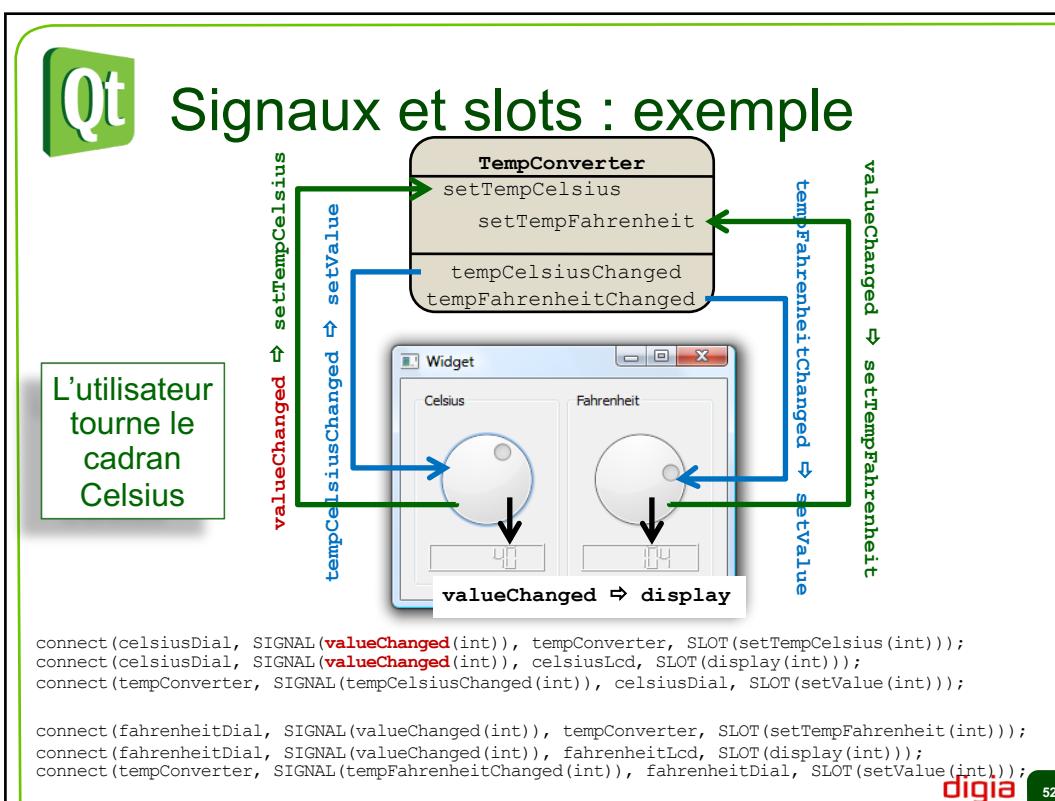
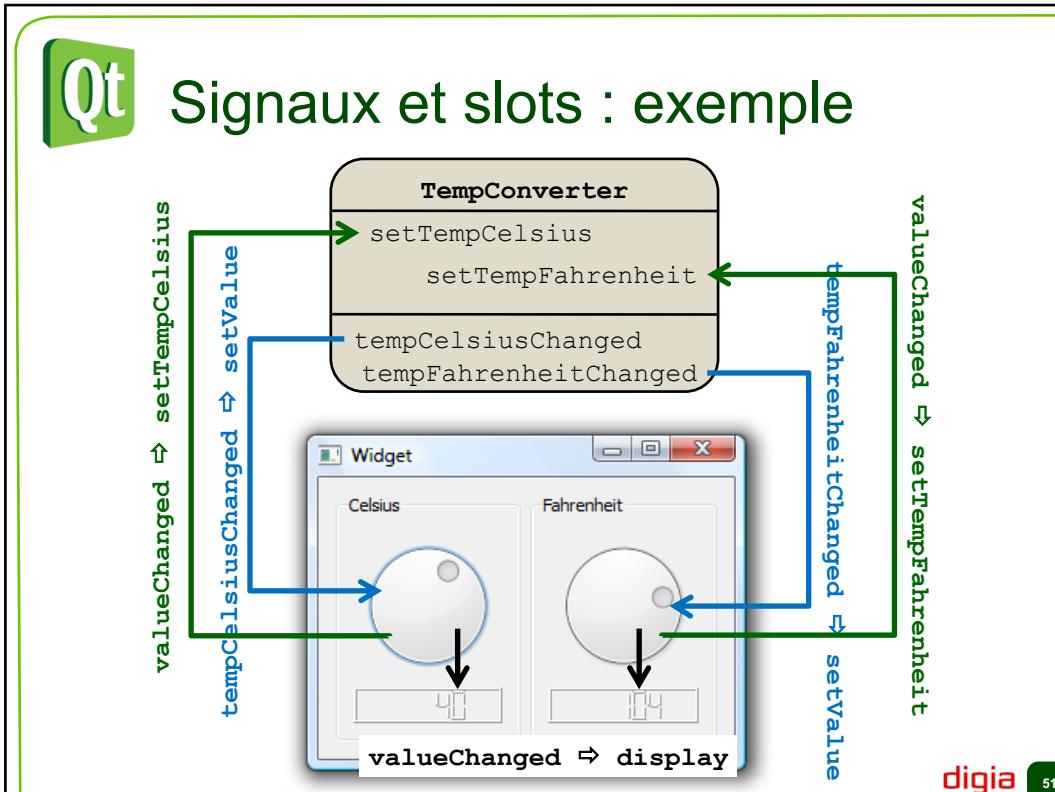
connect(fahrenheitDial, SIGNAL(valueChanged(int)),
        tempConverter, SLOT(setTempFahrenheit(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)),
        fahrenheitDial, SLOT(setValue(int)));
```

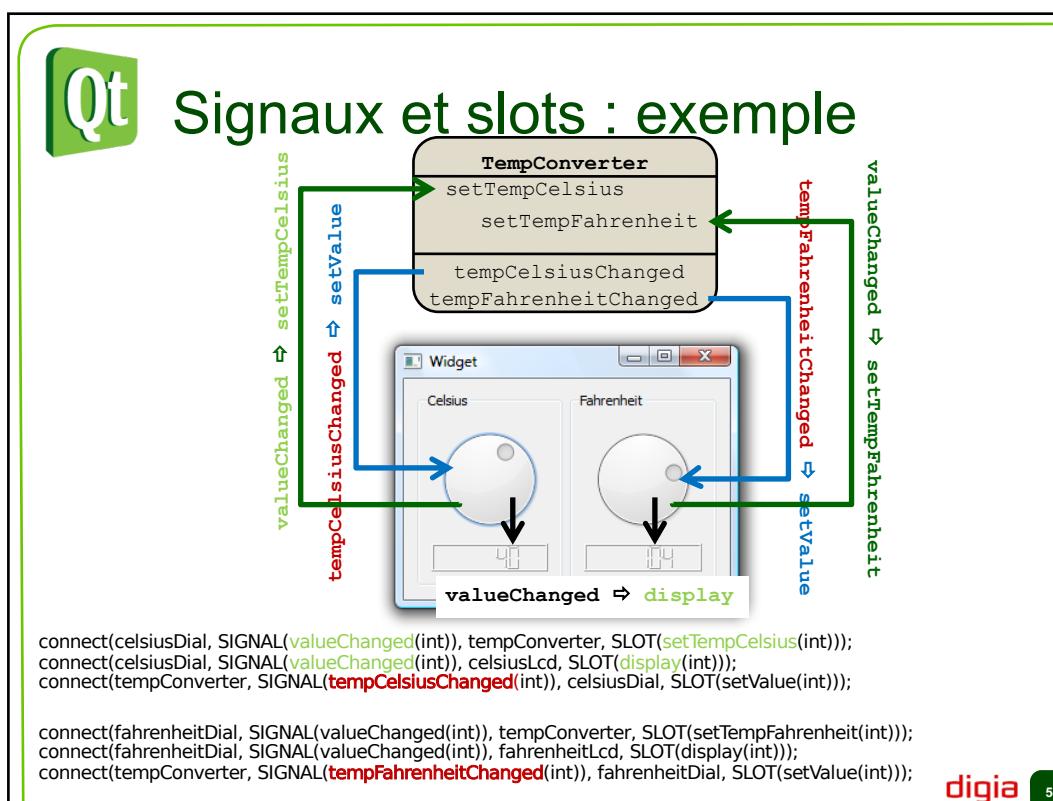
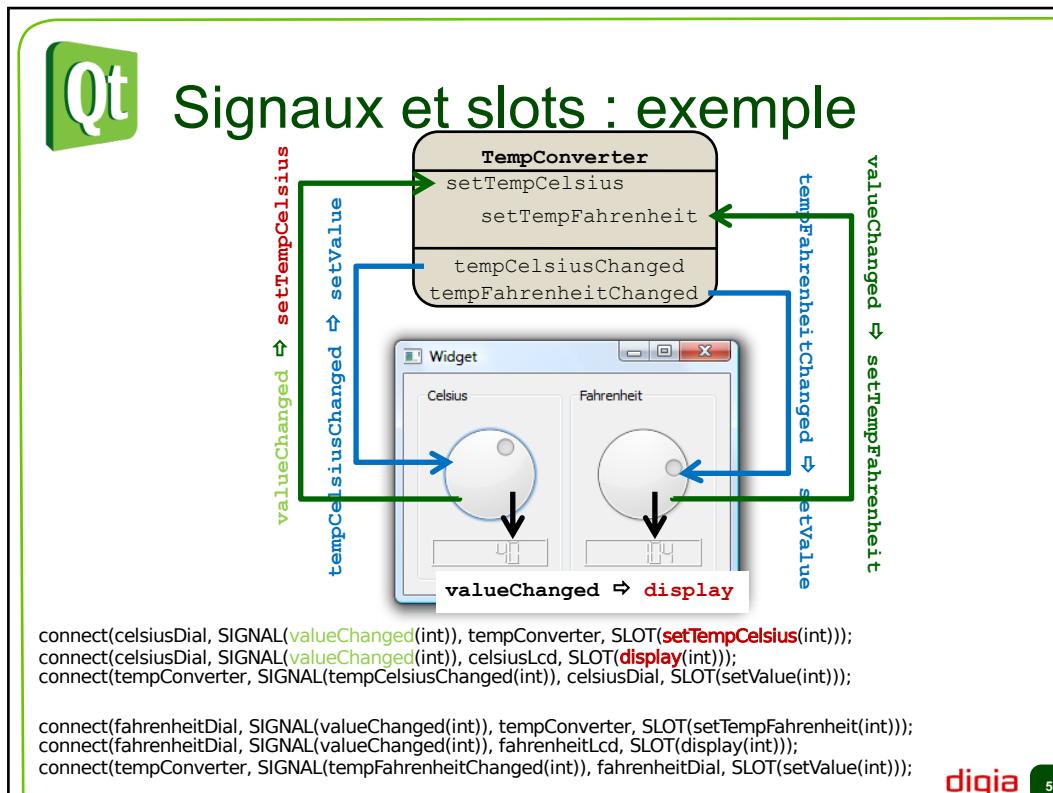
- Afficheurs LCD ⇒ gérés par les cadrans

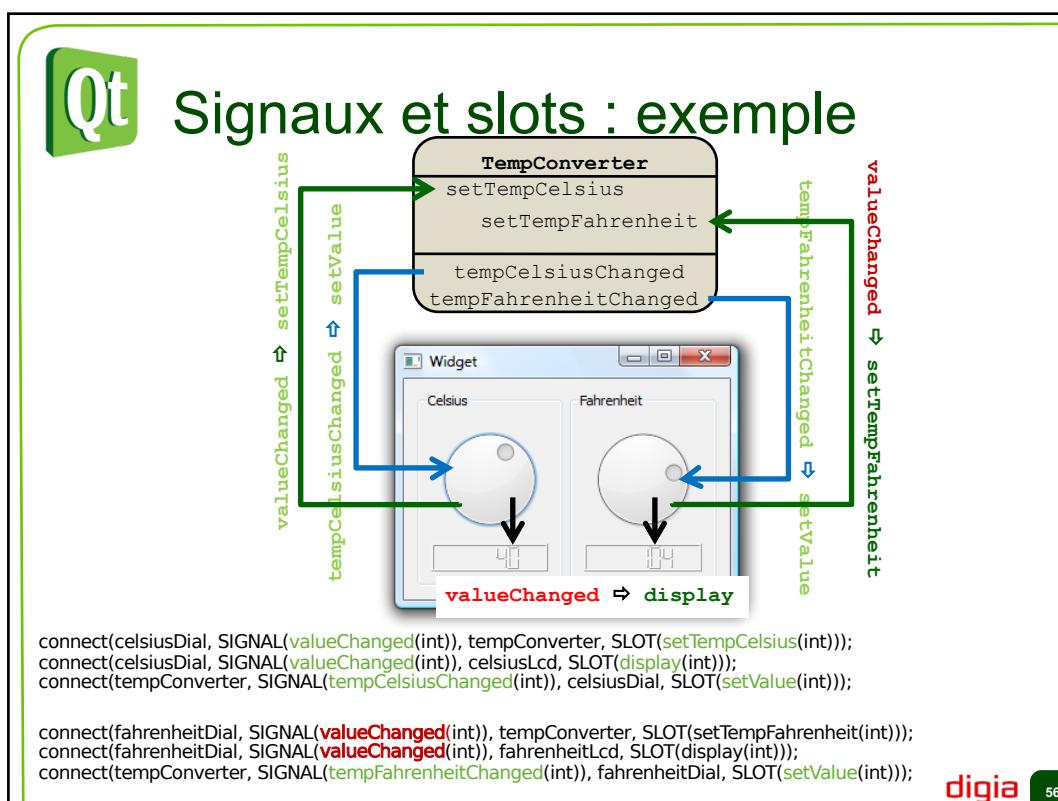
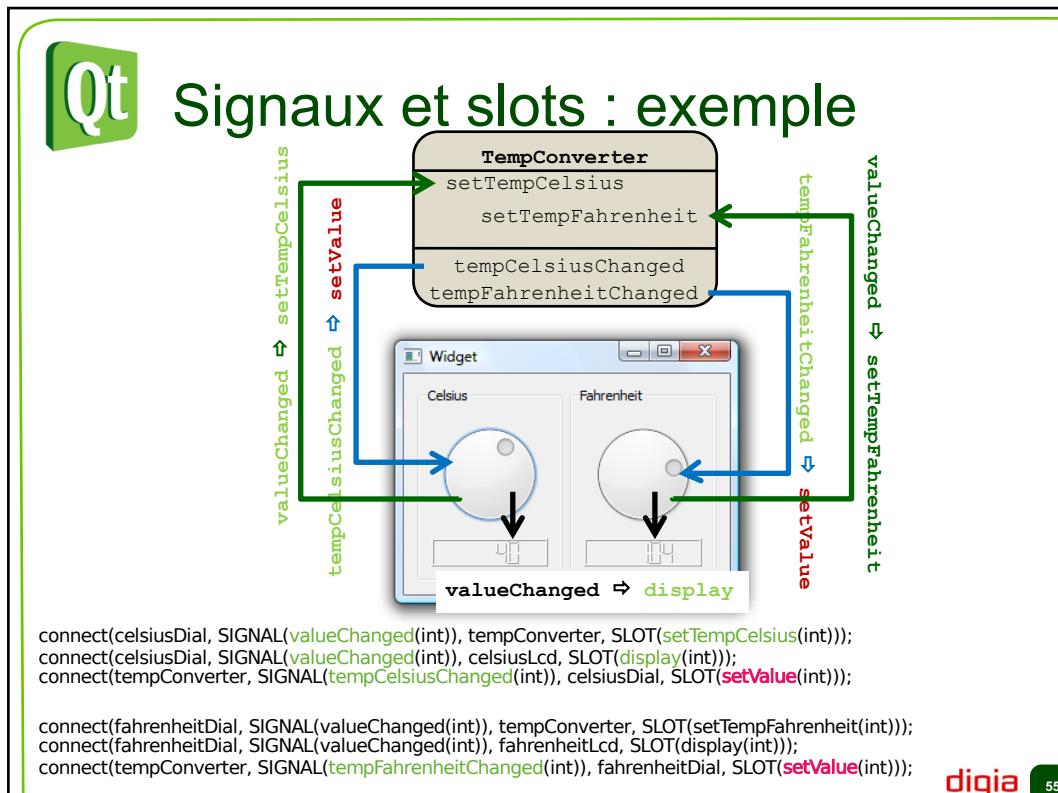
```
connect(celsiusDial, SIGNAL(valueChanged(int)),
        celsiusLcd, SLOT(display(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)),
        fahrenheitLcd, SLOT(display(int)));
```

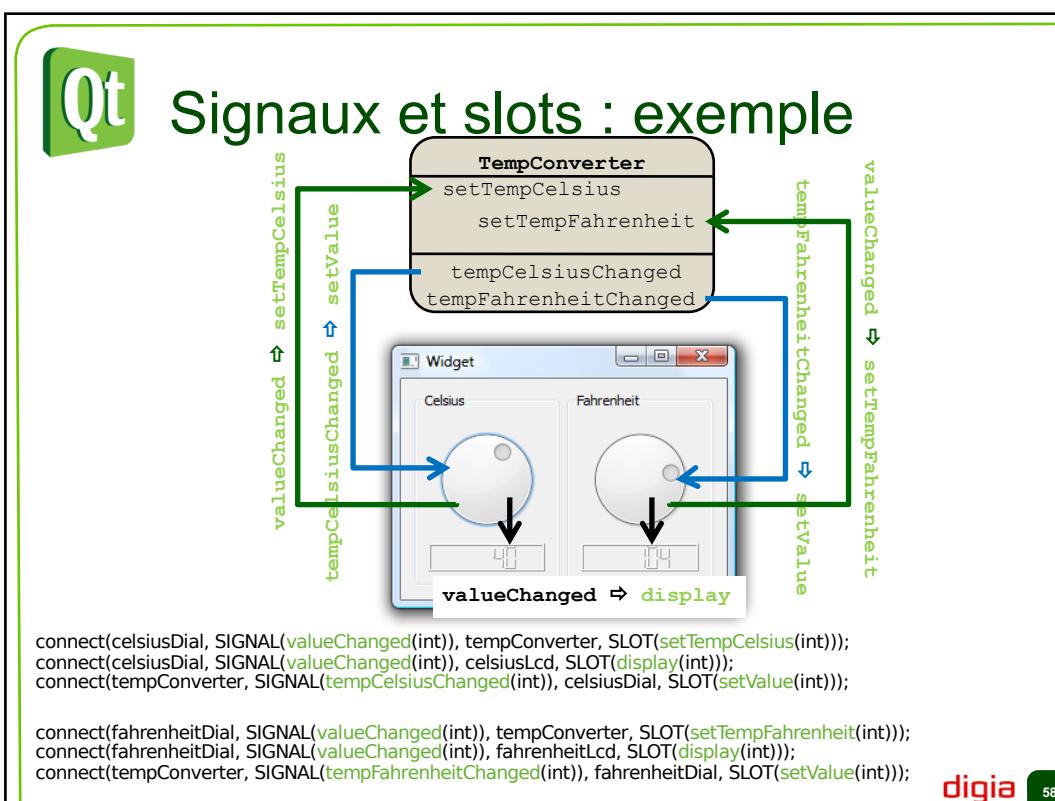
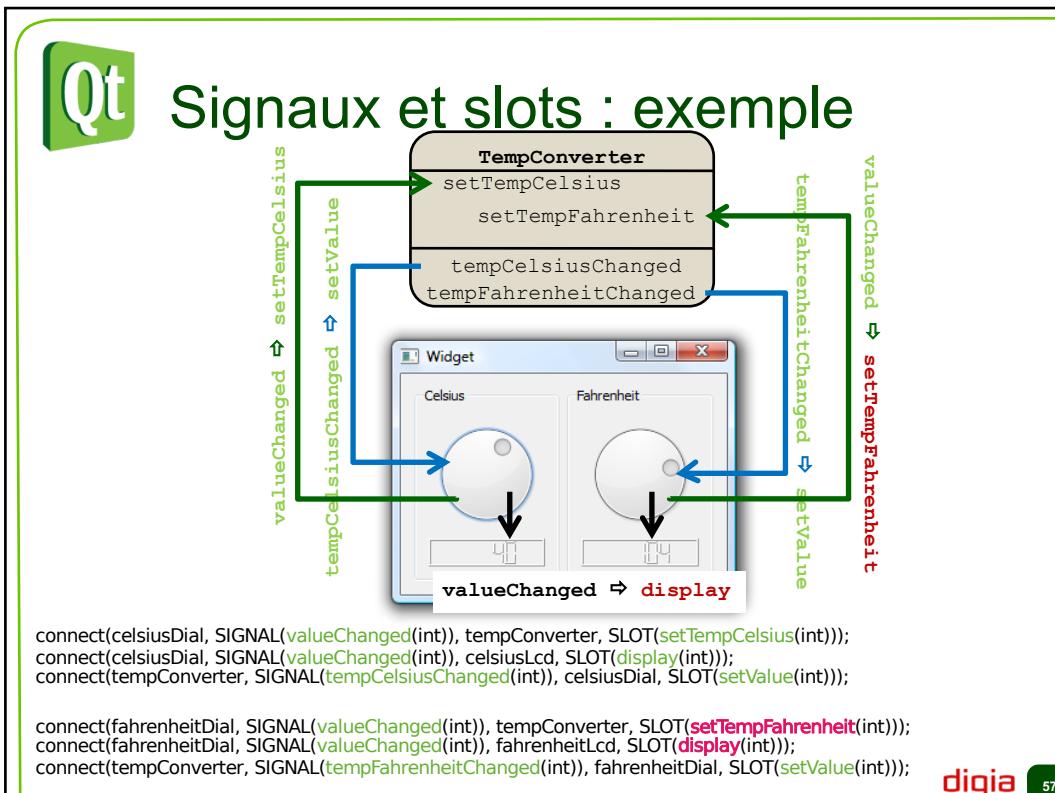
digia

50









Qt Connexion avec valeur?




- Comment transmettre une valeur à la connexion?
- Exemple : un clavier avec des QPushButtons



```
connect(key, SIGNAL(clicked()), this, SLOT(keyPressed(1)));
```

Impossible de passer une valeur !!!

digia 59

Qt Connexion avec valeur?



- Solution #1 ⇒ plusieurs slots



Connexions →

```
{
    ...
public slots:
    void key1Pressed();
    void key2Pressed();
    void key3Pressed();
    void key4Pressed();
    void key5Pressed();
    void key6Pressed();
    void key7Pressed();
    void key8Pressed();
    void key9Pressed();
    void key0Pressed();

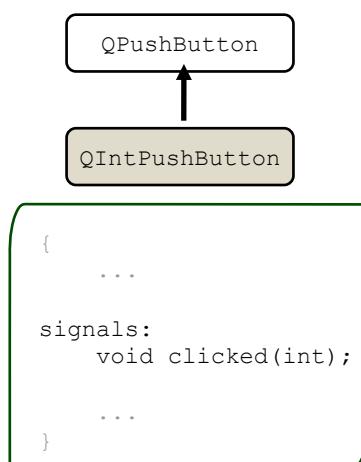
    ...
}
```

digia 60



Connexion avec valeur?

- Solution #2 ⇒ classe dérivée pour émettre signal



```

{
    QIntPushButton *b;

    b=new QIntPushButton(1);
    connect(b, SIGNAL(clicked(int)),
            this, SLOT(keyPressed(int)));

    b=new QIntPushButton(2);
    connect(b, SIGNAL(clicked(int)),
            this, SLOT(keyPressed(int)));

    b=new QIntPushButton(3);
    connect(b, SIGNAL(clicked(int)),
            this, SLOT(keyPressed(int)));
    ...
}
  
```

digia

61



Évaluation des solutions

- #1: plusieurs slots
 - Mais presque le même code
 - Difficile à maintenir
 - Petite modification ⇒ modifier tous les slots
 - Difficile à étendre : nouveau slot à chaque fois
- #2: nouvelle classe
 - Difficile à réutiliser
 - Difficile à étendre
 - une nouvelle classe pour chaque cas ?

digia

62



Solutions avancées

- #3: propriété dynamique et QObject::sender()
- Propriété ⇒ associer la valeur à l'objet

```
button1->setProperty(code, 1);
button2->setProperty(code, 2);
...
connect(button1, SIGNAL(clicked()), this, SLOT(keyPressed()));
```

- QObject::sender accès à l'objet émetteur

```
void keyPressed()
{
    ...this->sender()->code ...
}
```

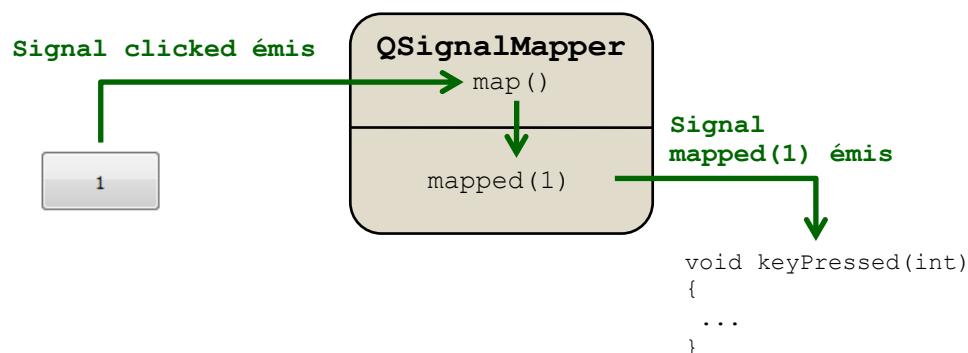
- Inconvénient : demande au slot de connaître l'émetteur

digia 63



Solutions avancées

- #4 : classe QSignalMapper
- Associe une valeur à chaque émission du signal clic



digia 64



Solutions avancées

- #4 : classe `QSignalMapper`
 - Associe une valeur à chaque émetteur

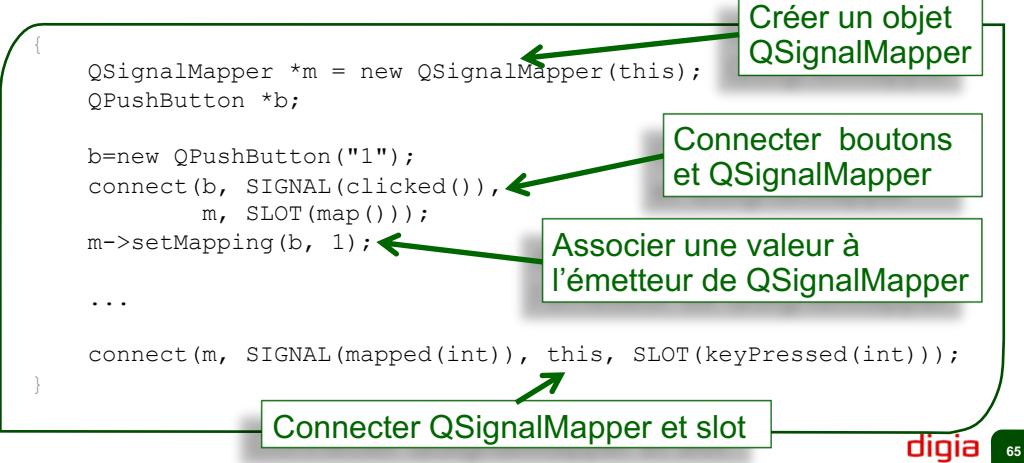
```
{  
    QSignalMapper *m = new QSignalMapper(this);  
    QPushButton *b;  
  
    b=new QPushButton("1");  
    connect(b, SIGNAL(clicked()),  
            m, SLOT(map()));  
    m->setMapping(b, 1);  
    ...  
  
    connect(m, SIGNAL(mapped(int)), this, SLOT(keyPressed(int)));  
}
```

Créer un objet `QSignalMapper`

Connecter boutons et `QSignalMapper`

Associer une valeur à l'émetteur de `QSignalMapper`

Connecter `QSignalMapper` et slot



digia 65