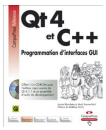
Qt dans l'Enseignement

Les événements

digia

Références bibliographiques

- Qt4 et C++ : Programmation d'interfaces GUI
  - Jasmin Blanchette
  - Mark Summerfield



- Traduction et adaptation supports enseignement

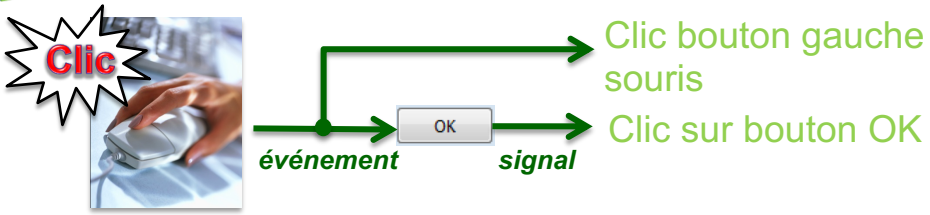
http://www.qt.io/training-materials/
http://www.qt.io/qt-essentials-widget-edition/

Gestion des événements

- Point de départ : QApplication::exec()
  - Lance la boucle d'attente des événements (event loop)
- Processus
  1. Génération des événements
     - Par les périphériques (souris, clavier, …)
     - Par Qt (les timers, …)
  2. File d'attente des événements ⇨ event loop
  3. Envoi des événements par QApplication vers Qobject
  4. Gestion des événements
     - Les QObject exécutent les réponses

digia

Événement vs signaux



- Signaux ⇨ déclenchés par widgets
  - Événements traduits car portent sur des widgets
- Événements ⇨ par système fenêtrage ou Qt
  - Liés
    - Au clavier, à la souris
    - Au raffraichissement,
    - À un chronomètre, etc.
  - Traités avant les signaux

digia

Événement vs signaux

- Utilité des événements?
  - Nouveau widget ⇨ pour décrire son comportement
  - Widget existant ⇨ pour modifier son comportement
- `QEvent` ⇨ classe de base
  - + de 100 types d'événements
    - `QMouseEvent` : classe dérivée pour événement souris
  - Méthode QEvent::type() retourne une énumération
    - `QEvent::MouseButtonPress` ⇨ événement bouton souris

digia

Mécanisme

- Événements ⇨ postés dans file d'attente de Qt
- Dans la file ⇨ **peuvent** être traités
  - Seul le dernier pour `mouseMoveEvent`
  - Demandes de raffraichissement ⇨ fusionnées
- `QObject` reçoit événement ⇨ `event` activée

```
bool QObject::event(Qevent *e)
```

  - Peut accepter (true retourné) ou ignorer l'événement
  - Si ignoré ⇨ propagation à travers la hiérarchie
  - Transmet événement aux gestionnaires d'événement
  - À surcharger pour intervenir avant ces gestionnaires

digia

Mécanisme

- Gestionnaire d'événements

```
void QWidget::mousePressEvent(QMouseEvent * event )
Void QWidget::keyPressEvent(QKeyEvent *event)
```

  - Défini au niveau de `QWidget`
  - Appelé par méthode `event`
  - Surchargé dans classe dérivée si autre comportement
  - À surcharger pour définir nouveaux comportements

Filtre d'événements

- But : intercepter des événements sur un `QObject`
- Filtre = `QObject`
  - Doit implémenter méthode `eventFilter`
  - Reçoit l'objet à observer
  - Autorise ou non ses événements
- Intérêt
  - Ajout de fonctionnalités sans créer classe dérivée
  - Empêcher un événement d'atteindre sa cible
- Filtre pour `QApplication`
  - Utile pour le débogage ou cas widgets désactivées

Filtre d'événements

- Plusieurs filtres possibles pour un objet
- Activation à tour de rôle
- Ordre : du dernier au premier installé

Filtre d'événements : exemple

- Filtre pour gérer nouvelle fonction sur les jauges
  - But : '0' appuyé ⇨ valeur = 0

```
class KeyboardFilter : public QObject
...
bool KeyboardFilter::eventFilter(QObject *o, QEvent *ev)
{
    if (ev->type() == QEvent::KeyPress)
        if (QKeyEvent *ke = static_cast<QKeyEvent*>(ev))
            if (ke->key() == Qt::Key_0)
                if (o->metaObject()->indexOfProperty("value") != -1 )
                {
                    o->setProperty("value", 0);
                    return true;
                }

    return false;
}
```

On s'assure que la propriété value existe pour l'objet o

Indique que l'événement est traité i.e. il ne sera pas passé à l'objet observé

Filtre d'événements : exemple

- Appliquer filtre ⇨méthode `installEventFilter`

```
ComposedGauge compg;
CircularGauge circg;

KeyboardFilter filter;

compg.installEventFilter(&filter);
circg.installEventFilter(&filter);
```

Objets filtrés

Objet qui filtre

- Ce filtre peut être utilisé pour différentes classes
  - QSlider, QDial, QSpinBox, etc.
  - Car n'utilise que propriété "value"

digia

Gestion des événements

- Traiter un événement : `event->accept()`
- Ignorer un événement : `event->ignore()`
- Propagation des événements
  - Se produit si l'événement est ignoré
  - Peut être propagé au parent du widget

digia 12

Gestion des événements

- Exemple
  - Evénement : QCloseEvent
  - Action : ferme la fenêtre si l'événement est accepté

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


Événement personnalisé

- Type dérivé de `QEvent`
  - `enum QEvent::Type` ⇨ numéro à définir
    - Doit être supérieur à `QEvent::User`
- Envoi géré par méthodes de `QCoreApplication`
  - `sendEvent()` ⇨ traité immédiatement
  - `postEvent()` ⇨ mis en attente sur la file
    - Traité au prochain tour de boucle d'attente des événements
    - Gestion optimisée car peut regrouper les mêmes événements