Qt 6 en pratique

écrit par Mériadeg Perrinel

Version imprimée 1.0

ERRATA

Page 82, §1

Au lieu de **a est l'enfant de b** :

Cet extrait de code montre comment placer un Rectangle a à droite d'un autre b. Comme b est l'enfant de a, son positionnement initial aurait dû être (x:0, y:0). Cependant, en le positionnant de la sorte, cela revient également à modifier son origine implicitement (x). C'est donc une autre manière d'arriver à vos fins. Comme la largeur de a vaut 20, qu'on demande au rectangle b d'être à droite de a, que **a est l'enfant de b** et que l'origine d'un composant est relative à son parent, alors b.x vaut ($a.x + a.width = 20$), ce qui se constate avec la sortie du programme QML : 20 0.

vous devez lire **a est le parent de b** :

Cet extrait de code montre comment placer un Rectangle a à droite d'un autre b. Comme b est l'enfant de a, son positionnement initial aurait dû être (x:0, y:0). Cependant, en le positionnant de la sorte, cela revient également à modifier son origine implicitement (x). C'est donc une autre manière d'arriver à vos fins. Comme la largeur de a vaut 20, qu'on demande au rectangle b d'être à droite de a, que **a est le parent de b** et que l'origine d'un composant est relative à son parent, alors b.x vaut ($a.x + a.width = 20$), ce qui se constate avec la sortie du programme QML : 20 0.

p. 83, §1

Au lieu de **rectangles bleus** :

Sur le motif le plus à droite, vous pouvez voir que les deux rectangles **bleus** sont également centrés horizontalement. Cela se fait via cette déclaration :

vous devez lire **rectangles** :

Sur le motif le plus à droite, les deux rectangles sont centrés horizontalement. Cela se fait via cette déclaration :

p.93, fin du 2^e \$

Au lieu de **txtInput2** :

touches du clavier, ici flèches droite ou gauche. Dans l'exemple, la largeur du composant identifié **txtInput2** augmente ou diminue.

vous devez lire **txtInput1** :

touches du clavier, ici flèches droite ou gauche. Dans l'exemple, la largeur du composant identifié **txtInput1** augmente ou diminue.

Dans les listings p. 118 et p. 120

Au lieu de :

```
if(data.m_Statistics != vector){
    data.m_Statistics = vector;
}
emit dataChanged(index, index, QVector<int>() << StatisticsRole
                << QuartilesRole);
emit quartilesChanged(QuartilesStats{data.m_Statistics}.toString());
```

vous devez lire (accolade déplacée) :

```
if(data.m_Statistics != vector){
    data.m_Statistics = vector;
    emit dataChanged(index, index, QVector<int>() << StatisticsRole
                    << QuartilesRole);
    emit quartilesChanged(QuartilesStats{data.m_Statistics}.toString());
}
```

p. 144, §1 après le titre

Au lieu de **C++** :

Faire d'une classe C++ un composant QML implique certaines règles à respecter, comme hériter de `QObject`, ajouter la macro `Q_OBJECT`, puis enregistrer l'objet et son module **pour pouvoir y accéder depuis le C++**. Ensuite, il est possible d'étoffer

vous devez lire **QML** :

Faire d'une classe C++ un composant QML implique certaines règles à respecter, comme hériter de `QObject`, ajouter la macro `Q_OBJECT`, puis enregistrer l'objet et son module pour pouvoir y accéder depuis le **QML**. Ensuite, il est possible d'étoffer

p. 226, listing

Au lieu de **IsTableExist** :

```
if((statisticsDataBase::IsTableExist(statisticsDataBase::statisticTableName))
{
    if(!statisticsDataBase::initializeStatisticsTable())
    {
        qCCritical(databaseController)<< tr("Unable to initialize statistic
table");
    }
}
```

vous devez lire **IsTableEmpty** :

```
if((statisticsDataBase::IsTableEmpty(statisticsDataBase::statisticTableName))
{
    if(!statisticsDataBase::initializeStatisticsTable())
    {
        qCCritical(databaseController)
        << tr("Unable to initialize statistic table");
    }
}
```

p. 229, \$2 et 3

Au lieu de **false** et **négatif** :

En effet, la commande SQL n'a pas pu s'exécuter correctement puisqu'elle demande la taille d'une table qui n'existe pas encore, la base de données étant vide à sa création. Ce fait permet de renvoyer **false** en retour de la méthode `isTableEmpty`.

Un retour **négatif** de la méthode `isTableEmpty` est la condition pour exécuter la méthode `initializeStatisticsTable` qui est responsable de l'ajout de la table `statistics` dans la base de données.

Vous devez lire **true** et **positif** :

En effet, la commande SQL n'a pas pu s'exécuter correctement puisqu'elle demande la taille d'une table qui n'existe pas encore, la base de données étant vide à sa création. Ce fait permet de renvoyer **true** en retour de la méthode `isTableEmpty`.

Un retour **positif** de la méthode `isTableEmpty` est la condition pour exécuter la méthode `initializeStatisticsTable` qui est responsable de l'ajout de la table `statistics` dans la base de données.

p. 287, listing

Au lieu de :

```
qCInfo(statisticsController) << "generateStatisticsAsync"
    << QThread::currentThread();
QFuture futureStats = QtConcurrent::run([numberOfValues]() {
    QList<double> res(numberOfValues, 0);
    ...
    return res;
}).then(QtFuture::Launch::Async, []{QList<double> generatedValues) {
    std::transform(generatedValues.begin(), generatedValues.end(),
        generatedValues.begin(), [](int value) {
            return value + 1;
        });
});
```

vous devez lire :

```
qCInfo(statisticsController) << "generateStatisticsAsync"
    << QThread::currentThread();
QFuture futureStats = QtConcurrent::run([numberOfValues]() {
    QList<double> res(numberOfValues, 0);
    ...
    return res;
}).then([]{QList<double> generatedValues) {
    std::transform(generatedValues.begin(), generatedValues.end(),
        generatedValues.begin(), [](int value) {
            return value + 1;
        });
});
```