

# Etendre le type QVariant

par Denys Bulant ([Tutoriels Qt](#))

Date de publication :

Dernière mise à jour :

Comment étendre QVariant avec ses propres objets ?  
Ce tutoriel va nous montrer comment étendre QVariant avec ses propres types.

## I - Introduction

## II - Exemples

### II-1 - Vector3

### II-2 - Utilisation de Vector3 avec QVariant

### II-3 - Centralisons un peu


### II-4 - Et les pointeurs ?

## III - Conclusion

## I - Introduction

**QVariant** est un type très puissant permettant de stocker n'importe quel type de variable sous une forme abstraite. Depuis Qt4, il est possible d'y rajouter ses propres types, ainsi que celui de librairies tierces. Il y a 2 restrictions :

- Connaître le nom du type (facile ;-) )
- Dans le cas d'une classe, elle doit répondre aux contraintes suivantes :
  - avoir un constructeur par défaut public (ne prenant aucun argument, ou tous les arguments ayant une valeur par défaut),
  - avoir un constructeur par copie public (c-à-d *MaClasse(const MaClasse&)* ),
  - avoir un destructeur public.

Il suffit d'ajouter la macro  **Q\_DECLARE\_METATYPE(type)** , après avoir inclus **QVariant** , et après la déclaration du type. Si le type à gérer est déclaré dans un en-tête que vous ne pouvez/voulez pas toucher (celui d'une bibliothèque tierce par exemple), il suffit de la saisir quelque part après l'inclusion dudit en-tête.

## II - Exemples

Voici un exemple illustrant cette fonctionnalité. Imaginons que nous voulions nous servir de **QVariant** pour transmettre les données d'un type *Vector3* défini dans une bibliothèque tierce (bien que les sources soient fournis ici pour éviter les multiples projets...).

### II-1 - Vector3

Ce type est défini dans le fichier suivant *vector3.h*.

```
vector3.h
#ifndef VECTOR3_H
#define VECTOR3_H
class Vector3
{
public:
    Vector3() : m_x(0.0), m_y(0.0), m_z(0.0) {}
    Vector3(float x, float y, float z) : m_x(x), m_y(y), m_z(z) {}
    Vector3(const Vector3 &v) : m_x(v.m_x), m_y(v.m_y), m_z(v.m_z) {}
    ~Vector3() {}

    float getX() const { return m_x; }
    float getY() const { return m_y; }
    float getZ() const { return m_z; }

    void setX(float x) { m_x = x; }
    void setY(float y) { m_y = y; }
    void setZ(float z) { m_z = z; }

private:
    float m_x, m_y, m_z;
};
// S'il s'agissait de votre propre classe et que vous vouliez la coupler pour de bon à Qt,
// vous voudriez très probablement ajouter ces 2 lignes:
// #include <QVariant>
// Q_DECLARE_METATYPE(Vector3);
#endif
```

### II-2 - Utilisation de Vector3 avec QVariant

Pas grand chose à expliquer ci-dessous, la documentation citée en début de document illustre la conversion vers et à partir d'un QVariant avec type personnalisé.

```
#include <iostream>
#include <string>
#include <QVariant>
#include "vector3.h"
// Nous enregistrons le type Vector3 pour son utilisation avec QVariant
// seulement nécessaire si ce n'est pas déjà dans le fichier d'en-tête
Q_DECLARE_METATYPE(Vector3);
using namespace std;
inline void displayVector3(const string &name, const Vector3 &v)
{
    cout << name << "={ "
         << v.getX() << ", "
         << v.getY() << ", "
         << v.getZ() << "}" << endl;
}
int main(int argc, char **argv)
{
```

```
Vector3 v1(1.0, 1.0, 0.5);
displayVector3(string("v1"), v1);
QVariant var;
var.setValue(v1); // var contient donc maintenant v1(1.0, 1.0, 0.5);
Vector3 v2(5.0, 0.75, 0.25);
displayVector3(string("v2a"), v2);
v2 = var.value<Vector3>(); // récupération dans v2 du Vector3 contenu dans var (ie v1)
displayVector3(string("v2b"), v2);
cin.get();
return 0;
}
```

## II-3 - Centralisons un peu

Voilà une première façon de faire. La récupération du **Vector3** contenu dans le **QVariant** se fait à l'aide d'une fonction template. Pour rendre les choses un peu plus lisible, et/ou si vous prévoyez de gérer un certain nombre de classes de cette façon, on peut tout simplement mettre ces macros à part, et regrouper les méthodes de conversion QVariant vers type personnalisé avec.

En voici un exemple :

```
#ifndef DECL_METATYPE_H
#define DECL_METATYPE_H
// Doit être inclus pour la macro Q_DECLARE_METATYPE et la définition de la classe Helper
#include <QVariant>
// Pour que Vector3 soit un type connu...
#include "vector3.h"
// Nous enregistrons le type Vector3 pour son utilisation avec QVariant
Q_DECLARE_METATYPE(Vector3);
// Namespace ajoutant des fonctions de conversion à partir de QVariant
namespace QVariantHelper
{
    inline Vector3 ToVector3(QVariant v)
    {
        return v.value<Vector3>();
    }
};
#endif
```

La seule modification à apporter au main serait la suivante :

```
[...]
Vector3 v2(5.0, 0.75, 0.25);
displayVector3(string("v2a"), v2);
// remplace l'utilisation d'un template par quelque chose
// de plus significatif (à mes yeux... ;)
v2 = QVariantHelper::ToVector3(var);
displayVector3(string("v2b"), v2);
[...]
```

## II-4 - Et les pointeurs ?

Quid des pointeurs vers des types personnalisés?

Eux aussi sont transformables en **QVariant** . Il suffit de les déclarer comme type eux aussi. En voici un exemple :

```
#ifndef DECL_METATYPE_H
#define DECL_METATYPE_H
#include <QVariant>
```

```
#include "vector3.h"
// Nous enregistrons le type Vector3 pour son utilisation avec QVariant
Q_DECLARE_METATYPE(Vector3);
// ... ainsi que le type pointeur vers Vector3
Q_DECLARE_METATYPE(Vector3*);
// Namespace ajoutant des fonctions de conversion à partir de QVariant
namespace QVariantHelper
{
    // renvoie le Vector3 contenu dans v, ou le Vector3 par défaut
    // si la conversion n'est pas faisable
    inline Vector3 ToVector3(QVariant v)
    {
        return v.value<Vector3>();
    }
    // renvoie le Vector3* contenu dans v, ou 0 si la conversion n'est pas faisable
    inline Vector3* ToVector3Ptr(QVariant v)
    {
        return v.value<Vector3*>();
    }
};
#endif
```

Et voici l'assignation d'un pointeur à **QVariant** et son extraction :

```
[...]
QVariant varVec3;
varVec3.setValue(v1);
QVariant varVec3Ptr = QVariant::fromValue(&v1); // Notation générique alternative pour
l'assignation d'une valeur à un QVariant
Vector3 v2(5.0, 0.75, 0.25);
displayVector3(string("v2a"), v2);
v2 = QVariantHelper::ToVector3(varVec3);
displayVector3(string("v2b"), v2);
Vector3 *v3 = QVariantHelper::ToVector3Ptr(varVec3Ptr);
// TOUJOURS vérifier si v3 est valide avant utilisation !
if(v3)
    displayVector3(string("v3(ptr)"), *v3);
else
    cout << "v3 is an invalid pointer to Vector3!" << endl;
[...]
```

Et enfin, si votre type est une classe dérivant de *QObject*, vous devez utiliser *QPointer* : en effet, *QObject* n'a pas de constructeur par copie.

Pour cela, il faut juste déclarer le metatype avec le *QPointer* représentant votre classe (ie `Q_DECLARE_METATYPE(QPointer<MyObject>)`) en plus des déclarations de *MyObject* et *MyObject\**.



***QPointer<VotreClasse> et VotreClasse\* sont incompatibles !***

### III - Conclusion

L'article original est situé ici :  **Etendre le type QVariant** .

Si vous avez des questions, n'hésitez pas à vous rendre sur **le forum Qt de Developpez** .

