

QSystemTrayIcon : ajoutez votre programme à la zone de notification avec Qt

par Alp Mestan ([Site perso de Alp](#)) ([Blog](#))

Date de publication :

Dernière mise à jour :

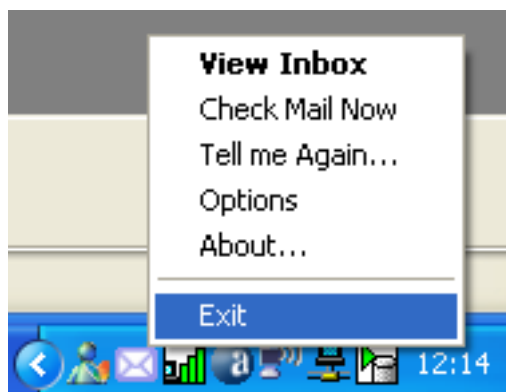
Cet article a pour but de vous présenter la classe QSystemTrayIcon, qui permet d'ajouter facilement un programme à la zone de notification.

- I - La zone de notification
- II - Utilisation de QSystemTrayIcon dans un programme
 - II-A - Notre programme
 - II-B - La classe QSystemTrayIcon
 - II-C - Ajout d'un icône dans la zone de notification.
- III - Conclusion

I - La zone de notification

De nos jours, beaucoup de programmes utilisent les icônes dans la zone de notification. Cela peut être dû à diverses raisons :

- le programme tourne en fond mais doit pouvoir communiquer avec l'utilisateur du système ;
- le programme est exécuté dans une fenêtre visible par l'utilisateur mais l'on veut pouvoir, par exemple, le réduire (le rendre visible uniquement dans la zone de notification), ou accéder à des options ou commandes rapidement ;
- ...



Zone de notification : Gmail Notifier tourne en fond et vous maintient informé de vos nouveaux messages sur votre compte gmail, configurable en quelques clics grâce à la zone de notification

C'est pourquoi la plupart des bibliothèques de création d'interfaces graphiques proposent une classe qui permet d'ajouter une icône dans la zone de notification pour son programme.

Qt étant une bibliothèque reconnue comme puissante et complète, elle se doit de fournir au développeur un tel outil. Cet outil, c'est la classe **QSystemTrayIcon**.

Dans cet article, nous allons voir comment rajouter un icône dans la zone de notification pour un programme. Nous allons donc dans la section suivante créer un programme simple, puis nous verrons comment fonctionne la classe **QSystemTrayIcon** et enfin nous ajouterons notre programme à la zone de notification. Il faut cependant savoir que **QSystemTrayIcon** est fournie avec Qt depuis la version 4.2.

II - Utilisation de QSystemTrayIcon dans un programme

II-A - Notre programme

Il nous faut avant toute chose un programme sur lequel nous baser pour montrer l'intégration d'un *QSystemTrayIcon*. C'est pourquoi nous allons utiliser le code suivant comme programme de base.

window.hpp

```
#ifndef WINDOW_HPP
#define WINDOW_HPP

#include <QSystemTrayIcon>
#include <QWidget>

class QLabel;

class Window : public QWidget
{
public:
    Window();
    ~Window();

private:
    QLabel* text;
};

#endif
```

window.cpp

```
#include "window.hpp"
#include <QLabel>

Window::Window()
{
    text = new QLabel("Hello World", this);
    resize(300,100);
}

Window::~Window()
{
    // Pas besoin de détruire (avec delete) text car en lui ayant fourni "this" (une Window)
    // comme parent, Qt s'occupe de transmettre la responsabilité de la destruction de
    // text au parent, ici notre instance de Window.
}
```

```
#include <QApplication>
#include "window.hpp"

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

Le programme ci-dessus ouvre une fenêtre contenant un simple texte, "Texte 0". L'objectif est ici de rajouter un icône pour notre programme dans la zone de notification, et de créer un menu accessible à partir de cet icône permettant de modifier le texte. Nous allons voir comment nous pouvons réaliser cela.

II-B - La classe QSystemTrayIcon

Il ne s'agit pas ici de décrire la classe dans son intégralité, comme le fait la doc de Qt, mais plutôt de mettre en avant ce qui va nous servir.

Tout d'abord, nous pouvons attacher un menu contextuel à un `QSystemTrayIcon`, à l'aide de la fonction membre `QSystemTrayIcon::setContextMenu(QMenu*)`. Celui-ci sera affiché lors d'un click droit sur l'icône.

Nous nous servirons également de la fonction membre `QSystemTrayIcon::show()` qui permet d'afficher notre `QSystemTrayIcon`.

Nous allons également utiliser `QSystemTrayIcon::setIcon(const QIcon&)` pour définir l'icône que nous allons utiliser.

Pour terminer, nous utiliserons `QSystemTrayIcon::showMessage` pour afficher une infobulle attachée à notre icône

Comme il est dit plus haut, il s'agit simplement de préciser les fonctions que nous allons utiliser, et non pas de lister les fonctions membres de `QSystemTrayIcon`. Il est maintenant temps d'intégrer `QSystemTrayIcon` à notre application.

II-C - Ajout d'un icône dans la zone de notification.

Pour commencer, nous devons ajouter un attribut privé `QSystemTrayIcon*` à notre classe `Window`. Nous le ferons de la manière suivante.

Ajout d'un attribut QSystemTrayIcon*

```
QSystemTrayIcon* sticon;
```

Pour pouvoir écrire cela, nous devons soit inclure l'entête `<QSystemTrayIcon>` soit écrire une déclaration anticipée de la classe `QSystemTrayIcon`. Nous choisirons la deuxième solution pour une compilation plus rapide.

Déclaration anticipée de QSystemTrayIcon

```
// code à placer avant la déclaration de Window  
class QSystemTrayIcon;
```

Nous devons maintenant initialiser notre icône de notification dans le constructeur. Des explications sont données en commentaires dans le code suivant.

```
Window::Window()  
{  
    text = new QLabel("Texte 0", this);  
    resize(300,100);  
  
    sticon = new QSystemTrayIcon(this); // on construit notre icône de notification  
  
    // Création du menu contextuel de notre icône  
    QMenu* stmenu = new QMenu(this);
```

```
QAction* actTexte1 = new QAction("Texte1",this);
QAction* actTexte2 = new QAction("Texte2",this);

stmenu->addAction(actTexte1);
stmenu->addAction(actTexte2);

sticon->setContextMenu(stmenu); // On assigne le menu contextuel à l'icône de notification

QIcon icon("icon.png");

sticon->setIcon(icon); // On assigne une image à notre icône

sticon->show(); // On affiche l'icône
sticon->showMessage("Bonjour","Hello, world!"); // On affiche une infobulle
}
```

Compiliez et exécutez ce code : vous devriez voir l'icône, dans la zone de notification, ainsi qu'une infobulle. L'objectif est atteint. Rajoutons de plus un changement de texte lors de clics sur les actions du menu.

Nous devons tout d'abord insérer `Q_OBJECT` au tout début de la déclaration de la classe, car nous allons définir des slots.

```
class Window : public QWidget
{
    Q_OBJECT
    // ...
};
```

Nous allons définir deux slots privés, car leur utilisation est limitée à notre classe. Soient `changeTexte1` et `changeTexte2` ces slots, attachés respectivement aux deux actions correspondantes. Ils se déclarent ainsi dans le fichier `window.hpp`.

```
private slots:
void changeTexte1();
void changeTexte2();
```

Il faut maintenant les définir dans le fichier `window.cpp`.

```
void Window::changeTexte1()
{
    text->setCaption("Texte 1");
}


void Window::changeTexte2()
{
    text->setCaption("Texte 2");
}
```

Il nous reste maintenant plus qu'à connecter ces slots aux actions, puis utiliser `moc` pour générer le nécessaire (pour l'utilisation de `moc`, référez vous à la documentation de Qt). Voici la connexion de ces slots aux actions.

```
connect(actTexte1, SIGNAL(triggered()), this, SLOT(changeTexte1()));
connect(actTexte2, SIGNAL(triggered()), this, SLOT(changeTexte2()));
```

Grâce à la documentation, vous pourrez facilement faire une utilisation plus poussée, mais ce n'est pas l'objectif de cet article.

III - Conclusion

Nous avons donc intégré notre icône dans la zone de notification grâce à la classe *QSystemTrayIcon*. Ne pensez pas que les fonctionnalités de cette classe s'arrêtent là. Le but de ce tutoriel était de vous initier à son utilisation ou même de vous la faire découvrir. C'est pourquoi je vous invite à explorer les autres fonctionnalités de cette classe qui n'ont pas été exposées ici. Je vous donne au passage un lien indispensable :  **Documentation officielle de Qt** : .

