Débuter avec Plasma

Par

Date de publication : 5 mai 2011

Dernière mise à jour : 5 août 2011

Dans ce tutoriel, nous allons créer un plasmoïde simple. Sans aller dans des complexités monstres, nous nous contenterons d'un plasmoïde statique qui contiendra les éléments suivants :

- une image SVG ;
- une icône ;
- un sympathique petit texte.

Commentez

I - Le résultat..... 3
II - Le code..... 3
II-A - Le fichier .desktop..... 3
II-B - Le fichier d'en-tête..... 3
II-B-1 - QRectF boundingRect()..... 4
II-B-2 - void paintInterface(QRectF contentsRect)..... 4
II-C - Le véritable coeur du plasmoide..... 4
II-C-1 - K_EXPORT_PLASMA_APPLET (<name>, <class>)..... 5
II-C-2 - Plasma/Svg..... 5
II-C-3 - setBackgroundHints(DefaultBackground)..... 6
II-C-4 - La méthode init()..... 6
II-C-5 - hasFailedToLaunch()..... 6
II-C-6 - setFailedToLaunch(bool, QString)..... 6
II-C-7 - dataUpdated(const QString &source, const Plasma::DataEngine::Data &data)..... 6
II-C-8 - Déterminer la taille de l'applet et sa géométrie : geometry() et contentsRect()..... 7
II-D - Construction de l'applet, le fichier CMakeLists.txt..... 7
III - Tester l'applet..... 7
IV - Remerciements..... 8

I - Le résultat

II - Le code

II-A - Le fichier .desktop

Tout plasmoïde doit fournir un fichier .desktop qui indique à Plasma comment le démarrer et comment il s'appelle.

plasma-applet-tutoriel1.desktop

```
[[Desktop Entry]
Name=Tutoriel 1
Comment=Tutoriel Plasma 1
Type=Service
ServiceTypes=Plasma/Applet

X-KDE-Library=plasma_applet_tutoriel1
X-KDE-PluginInfo-Author=Bas Grolleman
X-KDE-PluginInfo-Email=bgrolleman@emendo-it.nl
X-KDE-PluginInfo-Name=tutoriel1
X-KDE-PluginInfo-Version=0.1
X-KDE-PluginInfo-Website=http://plasma.kde.org/
X-KDE-PluginInfo-Category=Examples
X-KDE-PluginInfo-Depends=
X-KDE-PluginInfo-License=GPL
X-KDE-PluginInfo-EnabledByDefault=true
```

Les variables les plus importantes sont **X-KDE-Library** et **X-KDE-PluginInfo-Name**. Elles constituent le lien entre une classe et Plasma. Sans elles, rien ne démarrera. Pour la variable **X-KDE-PluginInfo-Category**, voir **PIG** (en anglais).

II-B - Le fichier d'en-tête

Voici le fichier d'en-tête de l'exemple. Des commentaires ont été ajoutés dans le code pour plus de clarté.

plasma-tutoriel1.h

```
// On se protège contre les inclusions multiples
#ifndef Tutoriel1_HEADER
#define Tutoriel1_HEADER
// On a besoin des en-têtes d'applet Plasma
#include <KIcon>

#include <Plasma/Applet>
#include <Plasma/Svg>

class QSizeF;

// Définition de l'applet Plasma
class PlasmaTutoriel1 : public Plasma::Applet
{
    Q_OBJECT
    public:
```

plasma-tutoriel1.h

```
        // Constructeur et destructeur simples
        PlasmaTutorial1(QObject *parent, const QVariantList &args);
        ~PlasmaTutorial1();

        // La méthode paintInterface() dessine l'applet à l'écran
        void paintInterface(QPainter *painter,
                const QStyleOptionGraphicsItem *option,
                const QRect& contentsRect);
 void init();

    private:
        Plasma::Svg m_svg;
        KIcon m_icon;
};

// Cette commande fait le lien entre l'applet et le fichier .desktop
K_EXPORT_PLASMA_APPLET(tutoriel1, PlasmaTutoriel1)

#endif
```

II-B-1 - QRectF boundingRect()

La fonction boundingRect() indique à Plasma la taille effective du plasmoïde. Elle est importante, car on a besoin de savoir quelle surface est occupée à l'écran.

En cas de problème avec le plasmoïde qui laisse des pixels derrière lui lorsqu'il est déplacé, ceci est habituellement dû à une implémentation incorrecte de boundingRect().

II-B-2 - void paintInterface(QRectF contentsRect)

contentsRect() peut être considérée comme la fonction principale, puisqu'elle dessine le plasmoïde à l'écran. C'est ici que l'on définit l'apparence du plasmoïde. On ne devrait dessiner que dans les limites définies par **contentsRect()** et éviter d'utiliser **geometry()**. Lorsqu'un plasmoïde n'a pas d'arrière-plan standard, par exemple s'il a été désactivé par un appel à setBackgroundHints() ou s'il se trouve dans le panel, geometry() et boundingRect() se comportent de la même manière. Cependant, lorsque l'arrière-plan standard est activé (ce qui est habituellement le cas), l'applet sera dotée d'une bordure dans laquelle elle ne doit pas dessiner.

II-C - Le véritable coeur du plasmoide

Voici le corps de la fonction, là aussi largement commentée.

plasma-tutoriel1.cpp

```
#include "plasma-tutorial1.h"
#include <QPainter>
#include <QFontMetrics>
#include <QSizeF>

#include <plasma/svg.h>
#include <plasma/theme.h>

PlasmaTutoriel1::PlasmaTutorial1(QObject *parent, const QVariantList &args)
    : Plasma::Applet(parent, args),
    m_svg(this),
    m_icon("document")
{
    m_svg.setImagePath("widgets/background");
    // Ceci amène l'arrière-plan standard sur un plateau, gratuitement !
```

plasma-tutoriel1.cpp

```
    setBackgroundHints(DefaultBackground);
    resize(200, 200);
}


PlasmaTutoriel1::~PlasmaTutoriel1()
{
    if (hasFailedToLaunch()) {
        // Faire du nettoyage ici
    } else {
        // Sauvegarde des paramètres
    }
}

void PlasmaTutoriel1::init()
{

    // Une petite démonstration de l'utilisation de la fonction '''setFailedToLaunch'''
    if (m_icon.isNull()) {
        setFailedToLaunch(true, "Personne à qui dire bonjour");
    }
}


void PlasmaTutoriel1::paintInterface(QPainter *p,
        const QStyleOptionGraphicsItem *option, const QRect &contentsRect)
{
    p->setRenderHint(QPainter::SmoothPixmapTransform);
    p->setRenderHint(QPainter::Antialiasing);

    // Maintenant, on dessine l'applet, en commençant par le SVG
    m_svg.resize((int)contentsRect.width(), (int)contentsRect.height());
    m_svg.paint(p, (int)contentsRect.left(), (int)contentsRect.top());

    // On place l'icône et le texte
    p->drawPixmap(7, 0, m_icon.pixmap((int)contentsRect.width(),(int)contentsRect.width()-14));
    p->save();
    p->setPen(Qt::white);
    p->drawText(contentsRect,
                Qt::AlignBottom | Qt::AlignHCenter,
                "Hello Plasmoid!");
    p->restore();
}

#include "plasma-tutoriel1.moc"
```

II-C-1 - K_EXPORT_PLASMA_APPLET (<name>, <class>)

Cette simple ligne est très importante, car elle établit le lien entre le nom de la classe et le nom de l'applet dans le fichier .desktop. Si l'applet semble ne pas se charger, cela peut être dû à une différence entre cette déclaration et le fichier .desktop.

K_EXPORT_PLASMA_APPLET ajoute "plasma_applet_" devant le nom. Pensez-y lorsque vous remplissez votre fichier *.desktop* afin d'éviter une différence dans le nom.

II-C-2 - Plasma/Svg

Comme on peut le voir dans cet exemple, on utilise un objet Plasma::Svg. Quelques points importants sont à noter ici.

Tout d'abord, on utilise un chemin relatif, **widgets/background**, ce qui amène Plasma::Svg à utiliser Plasma::Theme pour localiser les données SVG. Bien que Plasma::Svg puisse charger n'importe quel fichier à partir d'un chemin absolu, utiliser des chemins relatifs au thème aussi souvent que possible permet de personnaliser l'apparence de Plasma. Chacun des plasmoïdes apparaît alors comme un ensemble homogène plutôt que comme un ensemble d'applications séparées et sans liens entre elles. On peut obtenir une liste des composants graphiques disponibles sur la **page de thème Plasma** (en anglais).

Dans les deux cas, Plasma::Svg peut être utilisée pour ne dessiner qu'un sous-ensemble du fichier SVG en lui passant l'identifiant d'un élément apparaissant dans le document SVG. Le fichier clock.svg en est un bon exemple. En l'ouvrant, on constate qu'il est constitué d'un arrière-plan, de trois aiguilles (heures, minutes et secondes) et d'un avant-plan (le verre). Grâce à sa capacité à placer tous les éléments dans un même fichier, le document SVG affiche une horloge. Ceci est bien plus pratique pour les artistes que de créer cinq fichiers séparés et de les imaginer superposés les uns aux autres et aussi bien plus performant, puisqu'un seul rendu SVG et une seule lecture de fichier sont nécessaires.

II-C-3 - setBackgroundHints(DefaultBackground)

Comme le dessin de l'arrière-plan est une fonction courante, il existe un moyen plus simple pour le faire. En ajoutant setBackgroundHints(DefaultBackground) dans le code, l'arrière-plan par défaut de Plasma sera dessiné derrière le plasmoïde, ce qui économise et du temps et du code, mais ce qui permet également d'offrir à l'utilisateur une présentation plus homogène.

II-C-4 - La méthode init()

Dans le constructeur de l'exemple, le dialogue avec Plasma se cantonne à lui parler de l'arrière-plan et du fichier de configuration. Mais on peut également fixer la taille initiale du plasmoïde. Une fois cela fait, Plasma s'occupera de tout changement de taille et on n'aura plus à s'en soucier. Dans la méthode init(), il faut initialiser tout ce qui doit l'être, comme lire et appliquer des données de configuration.

II-C-5 - hasFailedToLaunch()

Si, pour une raison quelconque, l'applet ne parvient pas à démarrer (la bibliothèque n'a pas pu être chargée, un pilote matériel nécessaire n'a pas été trouvé, etc.), cette méthode retourne **true**. L'utilisation de cette fonction donne une occasion à votre application de faire un peu de ménage avant de s'arrêter.

II-C-6 - setFailedToLaunch(bool, QString)

Lorsque l'applet ne parvient pas à démarrer, cette fonction permet d'informer Plasma en fournissant optionnellement une raison. Plasma affichera alors une interface d'erreur standard pour informer l'utilisateur de la situation et, à partir de ce moment, les méthodes dessinant votre applet à l'écran ne seront plus appelées. Si le plasmoïde devient plus complexe et dépend de facteurs multiples pour démarrer, cette fonction est la meilleure façon de faire le ménage en cas d'échec.

II-C-7 - dataUpdated(const QString &source, const Plasma::DataEngine::Data &data)

Si on souhaite se connecter aux DataEngines de Plasma, on peut implémenter la méthode dataUpdated() dans le plasmoïde. Lorsqu'un DataEngine est directement connecté à la sous-classe de l'applet, dataUpdated() sera appelée à chaque fois que le DataEngine enverra des données mises à jour.

II-C-8 - Déterminer la taille de l'applet et sa géométrie : geometry() et contentsRect()

Si l'applet a besoin de connaître quelles sont sa taille et sa géométrie, on utilise contentsRect() et contentsRect().size(). Il faut éviter d'appeler geometry() et size(), car ces deux fonctions ne prennent pas en compte la taille de la marge fixée par l'arrière-plan par défaut des applets, également éviter d'utiliser des coordonnées absolues pour positionner les éléments de l'applet, comme QPoint(0, 0) pour se référer au coin en haut à gauche de l'applet. À la place, il est préférable d'utiliser contentsRect().topLeft().

II-D - Construction de l'applet, le fichier CMakeLists.txt

Pour finir, pour assembler tous les éléments, il est nécessaire de compiler le tout. Pour indiquer à QMake ce qui doit aller à quel endroit, on utilise le ficher CMakeLists.txt.

Plus de détails concernant CMake.

```
# Le projet a besoin d'un nom, bien sûr
project(plasma-tutoriel1)

# Trouve les bibliothèques requises
find_package(KDE4 REQUIRED)
include(KDE4Defaults)

add_definitions (${QT_DEFINITIONS} ${KDE4_DEFINITIONS})
include_directories(
   ${CMAKE_SOURCE_DIR}
   ${CMAKE_BINARY_DIR}
   ${KDE4_INCLUDES}
   )

# On insère les sources de notre code ici
set(tutorial1_SRCS plasma-tutorial1.cpp)

# On s'assure que les fichiers vont au bon endroit
kde4_add_plugin(plasma_applet_tutorial1 ${tutorial1_SRCS})
target_link_libraries(plasma_applet_tutorial1
                      ${KDE4_PLASMA_LIBS} ${KDE4_KDEUI_LIBS})

install(TARGETS plasma_applet_tutorial1
        DESTINATION ${PLUGIN_INSTALL_DIR})

install(FILES plasma-applet-tutorial1.desktop
        DESTINATION ${SERVICES_INSTALL_DIR})
```

III - Tester l'applet

Si l'environnement de développement actuel diffère de l'environnement de test d'installation, on doit lancer CMake avec -DCMAKE_INSTALL_PREFIX=`kde-config --prefix`, puis lancer make. Si le test est réalisé avec succès, l'applet peut être installée en lançant sudo make install ou cette suite de commandes :

```
cp ./lib/plasma_applet_tutorial1.so $KDEDIR/lib/kde4
cp ./plasma-applet-tutorial1.desktop $KDEDIR/share/kde4/services/
kbuildsycoca4
```

Ainsi, les applications KDE auront connaissance des nouveaux fichiers .desktop. Pour tester l'applet, on peut utiliser le programme **plasmoidviewer** :

```
plasmoidviewer applet_name
```

On peut même visionner l'applet dans un petit bureau en utilisant la même application :

```
plasmoidviewer -c desktop applet_name
```

où **applet_name** est la valeur spécifiée dans .desktop pour la clé **X-KDE-PluginInfo-Name**.

Sinon, on peut redémarrer Plasma, l'applet sera ainsi affichée dans le navigateur des applets :

```
kbuildsycoca4
kquitapp plasma
plasma
```

Si cela ne fonctionne pas, il faut alors relancer la session KDE en se déloguant pour se reloguer ou tenter d'exporter KDEDIRS=/usr/local:'kde4-config --prefix' puis relancer kbuildsyscoca4.

IV - Remerciements

Merci à **Thibaut Cuvelier** et à **Claude Leloup** pour leur relecture !