

Intégrer des gadgets logiciels (ou widgets) avec QWebKit

par Kent Hansen

Date de publication : 20 mai 2008

Dernière mise à jour :

Cet article est une traduction d'un article écrit par Kent Hansen sur le blog officiel de Qt

Il y explique comment intégrer facilement des gadgets logiciels (*widgets*) dans vos fenêtres Qt avec **Qt WebKit**.

Vous pouvez trouver l'originale **ici** et le code source sur le repository svn : **svn://labs.trolltech.com/svn**

Le code source étant basé sur QScript, voici une version c++ écrite par **Mongaulois** :
Source C++

N'hésitez pas à laisser vos commentaires ou poser vos questions sur le forum : <http://www.developpez.net/forums/d647328/c-cpp/bibliotheques/qt/integrer-gadgets-logiciels-widgets-qwebkit/>

Intégrer un gadget logiciel dans votre application Qt 4.4 avec QWebKit..... 3

Intégrer un gadget logiciel dans votre application Qt 4.4 avec QWebKit

L'interface de programmation d'intégration de Qt WebKit de la version 4.4 comporte une fonctionnalité très sympa qui vous permet d'intégrer des gadgets logiciels dans un QWebView. J'ai donc d'abord eu l'idée d'utiliser cette fonctionnalité pour intégrer des canevas Qt Designer à l'aide de QUILoader pour charger le canevas et Qt Script pour saisir la logique du canevas. Après avoir passé environ une heure pour tout paramétrer, j'aimerais mettre en évidence cette partie comme étant une partie où "tout se met en place parfaitement" - où les toutes nouvelles options de la version 4.4 se combinent aux bonnes vieilles fonctionnalités de jadis pour produire un accomplissement synergique qui stimule le cerveau.

Côté HTML, j'ajoute un onglet comme ceci.

```
<object type="application/x-qtform" width="500" height="400">
  <param name="form" value="http://chaos.troll.no/~khansen/calculator.ui"/>
  <param name="script" value="http://chaos.troll.no/~khansen/calculator.js"/>
</object>
```

L'onglet identifie un canevas et un script que le plug-in téléchargera et utilisera. Ensuite, je crée une sous-catégorie QWebPluginFactory qui sera invitée par WebKit pour créer mon plug-in. D'ordinaire, j'écrirais les classes liées au plug-in dans C++, mais puisque le Qt Script Bindings Generator (générateur de liaisons script) prend forme à présent (et parce que je reste un indémodable mordu de scripts), j'ai tout rédigé en Qt Script. L'usine de plug-ins ressemble à ceci.

```
<pre>
function MyWebPluginFactory(parent)
{
    QWebPluginFactory.call(this, parent); // call base class constructor }

MyWebPluginFactory.prototype = new QWebPluginFactory();

MyWebPluginFactory.prototype.create = function(mimeType, url, argumentNames, argumentValues) {
    if (mimeType != "application/x-qtform")
        return null;

    var formUrl = getArgumentValue("form", argumentNames, argumentValues);
    var scriptUrl = getArgumentValue("script", argumentNames, argumentValues);
    if (formUrl == undefined)
        return null;

    return new MyWebPlugin(new QUrl(formUrl), new QUrl(scriptUrl));
}
</pre>
```

La fonction QWebPluginFactory::create() est réexécutée pour gérer le type de mime *application/x-qtform*. Les URLs du canevas et du script sont transférés vers la nouvelle instance My WebPlugin ; c'est le gadget logiciel qui est actuellement intégré dans l'affichage. Le MyWebPlugin agit pour télécharger le canevas et le script et s'initialise paresseusement à mesure que les données nécessaires deviennent disponibles. Voici l'exécution complète de MyWebPlugin.

```
<pre>
function MyWebPlugin(formUrl, scriptUrl, parent) {
    QWidget.call(this, parent); // call base class constructor

    this.initialized = false;
    this.formReply = this.downloadFile(formUrl, this.formDownloaded);
    this.scriptReply = this.downloadFile(scriptUrl, this.scriptDownloaded); }

MyWebPlugin.prototype = new QWidget();

MyWebPlugin.prototype.downloadFile = function(url, callback) {
```

```

    if (this.accessManager == undefined)
        this.accessManager = new QNetworkAccessManager();
    var reply = this.accessManager.get(new QNetworkRequest(url));
    reply.finished.connect(this, callback);
    return reply;
}

MyWebPlugin.prototype.formDownloaded = function() {
    var loader = new QUiLoader();
    this.form = loader.load(this.formReply);
    var layout = new QVBoxLayout(this);
    layout.addWidget(this.form, 0, Qt.AlignCenter);
    this.initialize();
}

MyWebPlugin.prototype.scriptDownloaded = function() {
    var stream = new QTextStream(this.scriptReply);
    this.script = stream.readAll();
    this.initialize();
}

MyWebPlugin.prototype.initialize = function() {
    if (this.initialized)
        return;
    if ((this.form == undefined) || (this.script == undefined))
        return;
    var ctor = eval(this.script);
    if (typeof ctor != "function")
        return;
    this.instance = new ctor(this.form);
    this.initialized = true;
}
</pre>

```

La nouvelle interface de programmation de gestion de réseau présentée dans Qt 4.4 (QNetworkAccessManager et ses amis) sert à télécharger les fichiers. (Avec un plug-in plus élaboré, vous voudriez certainement afficher les informations de progression pendant le téléchargement des données, en utilisant le signal QNetworkReply's downloadProgress().) A noter également que puisque QNetworkReply est un QIODevice, créer le canevas et préparer le script à la fin des téléchargements est d'une élégance infinie. Le "protocole" utilisé pour appliquer le script au canevas est le suivant : Le script est évalué et si le résultat est une fonction, cette fonction est appelée comme argument avec le canevas. La fonction accroche ensuite la fonctionnalité aux composants du canevas. Il est temps de créer un affichage et de l'essayer.

```

<pre>
var view = new QWebView();
view.settings().setAttribute(QWebSettings.PluginsEnabled, true);

var factory = new MyWebPluginFactory();
view.page().setPluginFactory(factory);

view.load(new QUrl("script-calculator.html")); view.show();

QCoreApplication.exec();
</pre>

```

Activez vos plug-ins, paramétrez l'usine à plug-ins, chargez votre page et c'est parti !

qs_eval

Home · All Classes · Main Classes · Grouped Classes · Modules · Functions


TROLLTECH

QtScript Calculator Example

Files:

- [script/calculator/calculator.js](#)
- [script/calculator/calculator.ui](#)
- [script/calculator/calculator2.js](#)
- [script/calculator/prototype.js](#)
- [script/calculator/main.cpp](#)
- [script/calculator/calculator.pro](#)
- [script/calculator/calculator.qrc](#)

In this simple [QtScript](#) example, we show how to implement the functionality of a calculator widget.



The program logic in this example is a fairly straight port of the logic in the C++ [Calculator Example](#). The graphical user interface is defined in a UI file.

Exemple : Calculatrice

J'ai pris le HTML pour la documentation de [Qt Script Calculator Example](#), qui contient une image de l'application en action et je l'ai remplacé par l'onglet de l'image avec un onglet d'objet qui charge le canevas et le script. Alors, au lieu de regarder une capture d'écran, vous pouvez en fait utiliser la calculatrice qui se trouve sur la page de documentation ; le canevas et le script peuvent être les mêmes que ceux utilisés par l'application de la calculatrice seule. Neat-o.

Si vous voulez vous amuser avec, le code se trouve dans le dossier d'exemples du générateur de liaisons Qt (WebKitPlugins.qs).