Ajouter des scripts à votre application Qt

Par Jonathan Courtois

Date de publication : 19 novembre 2009

Cet article est un résumé de la conférence « Scripting your Qt application », réalisée par Ken Hansen durant les Qt Developer Days 2009 à Munich.
N'hésitez pas à commenter cet article !

I - Qui est Ken Hansen ?

Kent Hansen est ingénieur logiciel chez Nokia dans le département de développement du framework Qt à Oslo. Il est titulaire d'un master d'informatique de l'université NTNU (*Norwegian University of Science and Technology*). Il travaille depuis 2005 sur Qt Script, l'intégration de Javascript dans Qt et sur le framework de la machine à états.

II - Pourquoi Qt Script ?

L'objectif de Qt Script est, comme son nom l'indique, d'embarquer des scripts dans une application Qt. La base est réalisée en C++ mais avec une interface à l'environnement de script. On est sur une approche plug-in avec de l'interactivité (console). L'exemple type est le navigateur web avec un script chargé comme une page web et le navigateur qui exporte l'arbre DOM dans la page. Il s'agit d'un langage basé sur le standard ECMA-262, comparable à JavaScript sans la partie DOM. Il utilise une syntaxe de type Java. Qt Script a été créé dans trois grands buts :

- Personnaliser vos applications à l'aide de scripts ;
- Automatiser des tâches de votre logiciel ;
- Étendre votre application en permettant un nouveau modèle de déploiement.

III - Utilisation basique de Qt Script

Objectif : écrire une application qui utilise Qt Script pour calculer le sens de la vie.
Créer un projet Qt Core Application et ajouter le module Qt Script. Vous aurez alors un fichier de projet semblable à celui-ci :

QtScript.pro
```
QT       += script
QT       -= gui
TARGET    = QtScript
CONFIG   += console
CONFIG   -= app_bundle
TEMPLATE  = app
SOURCES  += main.cpp
RESOURCES += ressources.qrc
```

III-A - Premières lignes en Qt Script

main.cpp
```
#include <QtCore/QCoreApplication>
#include <QtScript>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QScriptEngine engine;
    QScriptValue result = engine.evaluate("41+1");
    qDebug() << result.toNumber(); // 42

    return a.exec();
}
```

L'environnement minimum de Qt Script réside dans *QtScriptEngine*. *QScriptValue* permet, entre autres, de :

- Tester les types et effectuer des conversions ;
- Accéder aux propriétés ;
- Appeler des fonctions à partir du C++.

III-B - Tester les types et effectuer des conversions

```
QScriptEngine engine;
QScriptValue result = engine.evaluate("41+1");
//Test du type
if (result.isNumber())
{
    //Conversion en chaîne de caractère
    qDebug() << result.toString(); // "42"
}
```

III-C - Utilisation d'un fichier Javascript

Dans la suite des exemples, la partie script sera dans un fichier script.js, ajouté en ressource au projet et exécuté par le code suivant :

Lecture du fichier script.js

```
QScriptValue evaluateFile(QScriptEngine &engine, QString const &filename)
{
    QFile file(filename);
    file.open(QIODevice::ReadOnly);

    return engine.evaluate(file.readAll(),filename);
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QScriptEngine engine;
    QScriptValue result = evaluateFile(engine, ":/script.js");
    qDebug() << result.toNumber(); // 42

    return a.exec();
}
```

script.js

```
41+1
```

III-D - Accéder aux propriétés

Exemple de code permettant d'accéder aux propriétés d'un objet Javascript

```
QScriptEngine engine;
QScriptValue result = evaluateFile(engine, ":/script.js");
qDebug() << result.property("food").toString(); // "spaghetti"
```

script.js

```
({
    food: "spaghetti"
})
```

III-E - Appeler des fonctions Javascript à partir du C++

Exemple de code appelant une fonction Javascript sans argument

```
QScriptEngine engine;
QScriptValue result = evaluateFile(engine, ":/script.js");
QScriptValue function = result.property("funkyFunction");
qDebug() << function.call(result).toString(); // "I love spaghetti"
```

script.js
```
({
food : "spaghetti",
funkyFunction : function(){
    return "I love " + this.food;
}
})
```

Et maintenant, le même exemple, mais en ajoutant comme argument le verbe de la phrase :

Fonction avec argument
```
QScriptEngine engine;
QScriptValue result = evaluateFile(engine, ":/script.js");
QScriptValue function = result.property("funkyFunction");
qDebug() << function.call(result,QScriptValueList() << "dislike ").toString(); // "I dislike
 meatballs"
```

script.js
```
({
food : "meatballs",
funkyFunction : function(verb){
    return "I " + verb + this.food;
}
})
```

III-F - Lister les propriétés

Ajouter une propriété et lister toutes les propriétés chargées
```
QScriptEngine engine;
QScriptValue result = evaluateFile(engine, ":/script.js");
result.setProperty("Bar", 123);
QScriptValueIterator it(result);
while(it.hasNext()){
    it.next();
    qDebug() << it.name() << it.value().toString();
}
```

script.js
```
"food" "meatballs"
"funkyFunction" "function(verb) {
    return "I " + verb + this.food;
}"
"Bar" "123"
```

III-G - Manipulation des erreurs

Les exceptions sont des objets, ce qui permet de manipuler facilement les erreurs, comme dans l'exemple suivant :

Une erreur comme un objet
```
QScriptEngine engine;
QScriptValue result = evaluateFile(engine, ":/script.js");
if (result.isError())
{
    qDebug() << result.property("message").toString(); // "noSuchVariable is not defined"
}
```

script.js
```
noSuchVariable
```

III-H - Objets globaux

Qt script donne également la possibilité de définir des objets globaux permettant d'accéder aux propriétés de ces objets n'importe où :

globalObjet
```
QScriptEngine engine;
QScriptValue global = engine.globalObject();
global.setProperty("MeaningOfLife",42,QScriptValue::ReadOnly);
QScriptValue result = evaluateFile(engine, ":/script.js");
qDebug() << result.toNumber(); // 6.48074
```

script.js
```
Math.sqrt(MeaningOfLife);
```

IV - Utilisation avancée de Qt Script

Dans cette seconde partie, nous allons travailler sur une interface graphique. Votre fichier projet ressemblera à un projet Qt Application :

QtScript.pro
```
QT        += script
TARGET     = QtScript
TEMPLATE   = app
SOURCES   += main.cpp
RESOURCES += ressources.qrc
```

Votre fichier main.cpp ressemble maintenant au suivant (ajout d'un bouton de taille 200x200) :

main.cpp
```
#include <QtCore/QCoreApplication>
#include <QtGui>
#include <QtScript>

QScriptValue evaluateFile(QScriptEngine &engine, QString const &filename)
{
    QFile file(filename);
    file.open(QIODevice::ReadOnly);

    return engine.evaluate(file.readAll(),filename);
}

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QPushButton button;
    button.resize(200,200);

    return a.exec();
}
```

La création d'un objet avec Qt Script se fait de la manière suivante :

Objet Qt Script
```
QScriptValue objet = engine.newObject();
```

L'avantage de ce type d'objet, c'est qu'il peut être modifié en cours d'exécution *via* les propriétés et les signaux/slots.

IV-A - Modification des propriétés

La modification des propriétés d'un *QObject* passe par la création d'un *newQObject* prenant en paramètre le *QObject* en question :

Modification des propriétés d'un bouton
```
QScriptEngine engine;
QScriptValue global = engine.globalObject();
QScriptValue scriptButton = engine.newQObject(&button);
global.setProperty("button",scriptButton);
QScriptValue result = evaluateFile(engine, ":/script.js");
```

script.js
```
button.text = "Hello Qt Script!";
button.styleSheet = "background : red";
button.show();
```

On obtient alors un bouton avec le texte "Hello Qt Script!" sur fond rouge.

IV-B - Signaux et Slots

Il est tout à fait possible de connecter un signal et un slot avec Qt SCript en utilisant la fonction *qScriptConnect*. Ajouter les lignes suivantes aux codes C++ et javascript :

Connexion d'un signal C++ à un slot Javascript
```
QScriptValue handler = result.property("buttonClickHandler");
qScriptConnect(&button, SIGNAL(clicked()), scriptButton, handler);
```

script.js
```
buttonClickHandler = function(){
    button.styleSheet = "background : green";
};
button.clicked.connect(buttonClickHandler);
```

En cliquant sur le bouton, le fond change de couleur, du rouge au vert.

IV-C - QtScriptDebugger

Pour utiliser le débogueur de Qt Script, il est nécessaire d'ajouter l'utilisation du module *scripttools* dans le fichier projet :

QtScript.pro
```
QT       += scripttools
```

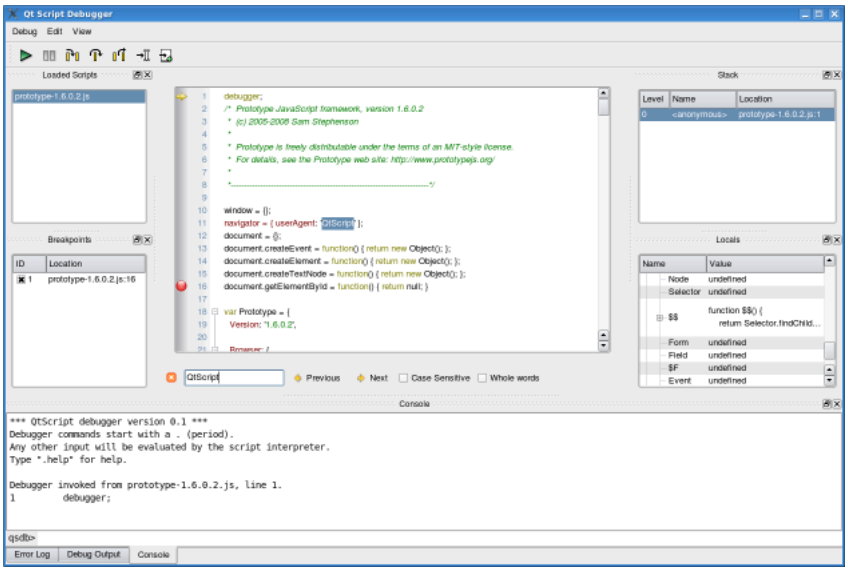
Ainsi que l'en-tête du module au début du fichier C++. Il suffit alors de créer un objet de type *QScriptEngineDebugger* et d'attacher l'objet *QScriptEngine* dessus.

main.cpp
```
#include <QtScriptTools>

...

QScriptEngineDebugger debugger;
debugger.attachTo(&engine);
```

Il est alors possible d'écrire et d'exécuter des scripts directement dans la console du débogueur.



Aperçu du débogueur Qt Script

V - Conclusion

Qt Script permet facilement de scripter une application Qt. Avec l'aide du débogueur, il est alors facile de maîtriser rapidement cette API. Pour plus d'informations : **QtScript/Generator sur Qt Labs**.

VI - Questions

Est-il possible d'ajouter des propriétés à des classes C++ ?
Oui, par le proxy, cela arrivera sous peu.
Est-il possible d'utiliser QtScriptDebugger dans Visual Studio ?
Non.
Est-il possible d'utiliser Qt Script dans des threads ?
Non, il n'est pas *thread safe*.
Est-il possible de limiter la mémoire utilisée par Javascript, et éviter ainsi tout crash ?
Non, ce n'est pas possible en dehors de la portée de Qt Script.