

Trolltech DevDays 2006 : Déploiement d'applications Qt

par Matthieu Brucher (<http://matthieu-brucher.developpez.com/>) (Blog)

Date de publication : 01/11/2006

Dernière mise à jour :

Comment déployer des applications Qt, résumé de la conférence d'Harald Fernengel lors des Trolltech DevDays 2006.

- I - Qui est Harald Fernengel ?
- II - Résumé de la présentation
 - II-A - Les questions non dépendantes d'une plateforme
 - II-A-1 - Les bibliothèques statiques
 - II-A-2 - Les bibliothèques dynamiques
 - II-A-3 - Les plug-ins et les ressources
 - II-B - Les problèmes spécifiques à une plateforme
 - II-B-1 - Windows
 - II-B-2 - Mac OS X
 - II-B-3 - Linux
 - II-C - Tweaks
- III - Conclusion

I - Qui est Harald Fernengel ?

Harald Fernengel est le responsable des releases de Qt, donc concerné au premier point par le déploiement des applications Qt.

II - Résumé de la présentation

Harald est parti du principe que tout le monde utilise **qmake**, pour lequel il n'existe pas de moyen de créer directement un moyen de créer un installateur. Lorsqu'on expédie son programme au format zip ou tar.gz, que faut-il inclure ?

II-A - Les questions non dépendantes d'une plateforme

Les questions indépendantes des plateformes concernent le choix statique ou dynamique et leur incidence sur les plugins et les ressources.

II-A-1 - Les bibliothèques statiques

Leur avantage immédiat est que l'application finale est monolithique, donc son déploiement est très simple, il suffit de copier l'exécutable sur la plateforme cible. De plus, des optimisations lors de l'édition des liens peuvent se produire, permettant d'accélérer le programme ou de réduire sa taille. En revanche, un fix sur le programme entraîne de gros patches. De même, l'édition des liens avec un Qt statique entraîne la nécessité d'inclure toutes les bibliothèques annexes à Qt, mais heureusement, **qmake** s'en charge.

II-A-2 - Les bibliothèques dynamiques

Qui dit bibliothèque dynamique dit plusieurs fichiers, un pour chaque bibliothèque au moins. Un avantage immédiat est qu'un patch n'entraîne la réexpédition que d'une partie du programme, et de plus toutes les bibliothèques peuvent utiliser la même version de Qt, la même bibliothèque dynamique. Par conséquent, on peut utiliser les plugins de manière dynamique - on n'utilise qu'une seule instance de Qt -, d'où flexibilité. En revanche, il faut penser à envoyer toutes les bibliothèques.

II-A-3 - Les plug-ins et les ressources

Depuis la version 4.2 de Qt, on peut charger dynamiquement les ressources d'un programme s'il est compilée avec la version dynamique de Qt. Une des ressources les plus utilisées est l'icône de l'application, qui peut être forcée à l'aide de **QWidget::setWindowIcon()**.

De même, on peut utiliser les Qt plug-ins, qui seront chargés à la demande - donc une empreinte mémoire potentiellement plus faible -, et le chemin d'accès aux plug-ins peut être définie par un fichier **qt.conf**, par la définition de **QT_PLUGIN_PATH** dans le projet ou directement dans l'application grâce à **QApplication::setLibraryPaths()** ou **QApplication::addSearchPath()**.

II-B - Les problèmes spécifiques à une plateforme

Chaque plateforme a ensuite ses spécificités.

II-B-1 - Windows

Déployer sous Windows implique de fournir **mingw10.dll** lors de la compilation avec MingW ou les bibliothèques **msvcr*.dll** et **msvc*.dll** pour Visual Studio. Pour Visual Studio 2005, il faut en plus ajouter **CONFIG += embed_manifest_exe** pour générer le manifeste. D'ailleurs Qt4.2 est la première version de la bibliothèque à inclure automatiquement le manifeste lors de sa compilation propre.

II-B-2 - Mac OS X

Sur cette plateforme, on peut définir un **application bundle**. L'application bundle est construit par défaut - si j'ai bien compris -, et donc peu de choses restent à faire par le programmeur. Par exemple l'ajout de l'icône de l'application se fait par un **ICON=theapp.icns**.

Pour générer un binaire universel - PPC + Intel -, il suffit de lancer **qmake** avec le flag **-universal**.

II-B-3 - Linux

Le gros problème des versions Linux est leur diversité. Heureusement, il existe LSB - Linux Standard Base - qui permet de définir ce qu'une distribution Linux fournit au minimum au niveau de l'interface. Qt3.3 en fait partie et Qt4.2 en est un module - FC5 ne fournit pas le module, et c'est la seule des distributions participants au LSB -. Heureusement, seul Trolltech fournit une implémentation de Qt.

A l'aide de **qmake -prefix op/theapp**, on peut définir l'endroit de l'installation du logiciel. Pratique si on ne veut pas se reposer sur le LSB. De plus qmake utilise rpath pour fixer le chemin d'accès.

En définitive, c'est assez simple.

II-C - Tweaks

Les plugins doivent être ouverts pour être catalogués, sauf si vous le faites vous-même en les plaçant dans un sous-dossier dont le nom est le type du plug-in - imageformats, sqldrivers, ... comme c'est le cas de Qt elle-même -.

On peut paramétrer **gcc** pour rendre la table des symboles plus petites -> **-fvisibility**, permettant de n'exporter que les symboles qu'on veut, le comportement pas défaut de Visual Studio, en fait.

On peut aussi ajouter le flag **-no-exceptions** si on n'utilise pas les exceptions, Qt ne les utilisant pas.

Il est possible de faire une analyse intermodules du code C à l'aide de **-combine**.

On peut configurer Qt pour cross-compiler du code 32bits sur un Linux 64bits car la chaîne de compilation existe en général, il faut ajouter un **-platform linux-g++32** à l'argument de configure dans ce cas.

III - Conclusion

Une conférence abordable par le plus grand nombre, un gros rappel de ce qu'est une bibliothèque et de comment les utiliser efficacement, avec leurs avantages et leurs inconvénients. Pour plus de détail, veuillez consulter [cette page](#).

