Table des matières

Liens....1
Objectif....1
Exemple : bonjour à tous !....2
  Mise en situation....2
  Déploiement....3
  Fabriquer une version finale de l'application (release)....3
  Fabriquer un paquetage pour l'application (rpmbuild)....4
  Déployer une application sur une machine (installation)....4
Fabriquer un paquetage....5
  RPM (Red Hat Package Manager)....5
Tâches préliminaires....6
  Installation requise....6
  Arborescence....6
  Configuration pour Mandriva Linux....7
  Vérifications préliminaires 7
Le fichier .spec....8
  Exemple de contenu d'un fichier .spec....10
  Description du contenu d'un fichier .spec....11
    Section En-tête....11
    Section Prep 13
    Section Build....14
    Section Install 14
    Section Files....15
    Section Changelog 16
La construction 17
  Installer le paquetage créé....18
  Ajout de l'application dans les menus KDE....18
  Tests....21
  Une alternative : checkinstall 22

Annexe 1 : Qt....23
Annexe 2 : Qmake....24

Liens

http://wiki.mandriva.com/fr/Construire_des_paquetages_RPM
http://wiki.mandriva.com/en/Development/Howto/XDGMenuSystem
http://www.gurulabs.com/downloads/GURULABS-RPM-LAB/GURULABS-RPM-GUIDE-v1.0.PDF
Notion de bibliothèque : http://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle
Max-rpm : http://www.rpm.org/max-rpm/

Objectif

Dans le cadre du déploiement d'une application, réaliser un paquetage d'installation de celle-ci sous Mandriva Linux.

Remarque : les commandes à exécuter pour le TP sont indiquées en **`gras`**.

© BTS IRIS 2010 <tv>

Exemple : bonjour à tous !

Mise en situation

L'exemple est le classique " Hello world ". Cette application a été réalisée avec l'environnement de développement QDevelop (voir Annexe 1 sur Qt).

L'application est disponible dans l'archive : **qhelloworld-1.0.0.tar.gz**.

Une fois l'application réalisée, pour créer cette archive, on a réalisé les étapes suivantes :

- modifier le fichier de projet **qhelloworld.pro** pour y intéger des fichiers supplémentaires :

```
TEMPLATE = app
QT = gui core
CONFIG += qt release warn_on console
DESTDIR = bin
OBJECTS_DIR = build
MOC_DIR = build
UI_DIR = build
FORMS = ui/qhelloworld.ui
HEADERS = src/qhelloworld.h
SOURCES = src/qhelloworld.cpp src/main.cpp
# Ajout pour fabrication RPM (2010 <tv>)
# Faire : qmake puis make dist pour créer l'archive .tar.gz
DISTFILES += AUTHORS COPYING NEWS ChangeLog INSTALL README TODO man/man1/*
extra.path = usr/share/qhelloworld
extra.files = AUTHORS COPYING NEWS ChangeLog INSTALL README TODO
man.path = usr/share/man/man1
man.files = man/man1/*
target.path = usr/bin
INSTALLS += target extra man
```

- créer le fichier **Makefile** en tapant : **qmake**

- modifier le fichier **Makefile** en exécutant le script : **./maketorpm.sh**

```
#!/bin/bash
# transforme les noms xxx1.0.0 en xxx-1.0.0 (2010 <tv>)
# nécessaire pour le respect de la convention Mandriva pour les RPMS
mv Makefile .Makefile
cat .Makefile | sed 's/1.0.0/-1.0.0/g' > Makefile
rm -f .Makefile
```

- pour finir, on crée l'archive en tapant : **make dist**

 © BTS IRIS 2010 <tv>

Déploiement

Pour réaliser une procédure d'installation d'une application, il faudra décomposer celle-ci en trois parties :

- fabriquer une version finale de l'application (make)
- fabriquer un paquetage pour l'application (rpmbuild)
- déployer cette application sur une machine à partir d'un installateur (rpm)

Ces trois parties sont parfois dépendantes de la plateforme et des outils utilisés.

Fabriquer une version finale de l'application (release)

De manière générale, il faut résoudre un certain nombre de problèmes et répondre à quelques choix.

<u>Droits ?</u>
Les droits et les fichiers à déployer (consulter le contrat de licence de Qt) et les droits et les fichiers à appliquer à son application (indispensable de lire http://www.gnu.org/licenses/license-list.fr.html).

<u>Statique ou dynamique ?</u>
Il faudra choisir entre fabriquer un exécutable indépendant (statique) ou non (dynamique). Sous Linux, les dépendances sont nombreuses : architecture multi-platerforme, librairies, etc … (sous Windows, la dépendance est le plus souvent assurée par des DLLs).

<u>Librairies dynamiques ?</u>
Aide : http://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle

Les librairies dynamiques contiennent du code exécutable (des fonctions formant une API) qui sera susceptible d'être utilisé par un (ou plusieurs) programmes au moment de leur exécution. En cas de besoin, la librairie dynamique sera chargée en mémoire et son code sera alors utilisable par le programme demandeur. Il en résulte les avantages suivants : la taille du programme est réduite (puisque le code dont il a besoin se trouve dans la librairie), la possibilité de faire évoluer la librairie sans avoir à recompiler (si le prototype des fonctions définies dans la librairie reste inchangé). L'inconvénient majeur reste l'obligation de la présence de la librairie sur le système cible pour que le programme puisse s'exécuter.

L'extension usuelle d'une librairie dynamique sous Linux est .so (pour Windows, ce sera .dll)

<u>Dépendances ?</u>
Il y a plusieurs techniques pour connaître les dépendances externes d'une application :

- utiliser un utilitaire qui recherche ces dépendances (la commande ldd sous Linux, la commande rpmbuild établit aussi une liste de dépendances de paquetages nécessaires, Dependency Walker sous Windows, etc.)

- appliquer une démarche (rudimentaire !) qui teste l'application sur une machine vierge et qui résout les dépendances les unes après les autres.

 © BTS IRIS 2010 <tv>

Les fichiers de l'application à déployer ?

Il faut évidemment recenser les fichiers (ainsi que l'arborescence) qui composent l'application et qui devront être déployer avec celle-ci. De manière générale, on trouve :

- l'exécutable (.exe sous Windows)
- l'icône (.ico)
- des fichiers de configuration (comme les .ini)
- des fichier multimédia (comme des images)
- des fichiers d'aide (comme les .chm ou les pages man, des fichiers .html, ...)
- un fichier licence, un fichier readme, ...
- etc ...

Exemple :

```
./AUTHORS
./ChangeLog
./COPYING
./INSTALL
./NEWS
./README
./TODO
./bin/qhelloworld
./man/man1/qhelloworld.1.lzma
```

Ici, la procédure de fabrication est simple :

```
$ make
$ make install
```

Elle sera réalisée directement dans l'étape de création du paquetage.

Fabriquer un paquetage pour l'application (rpmbuild)

Pour le déploiement de l'application, notre choix se porte sur les paquetages RPM.

La fabrication d'un paquetage est décrite dans le chapitre suivant.

Déployer une application sur une machine (installation)

Pour installer le paquetage (sous root) :

```
# rpm -ivh ./qhelloworld-1.0.0-1mdv2009.1.i586.rpm
Préparation...              ########################################### [100%]
   1:qhelloworld            ########################################### [100%]
```

Pour désinstaller le paquetage (sous root) :

```
# rpm -ev qhelloworld-1.0.0-1mdv2009.1.i586
```

Pour obtenir des infos sur le paquetage, exécuter le script fourni :

```
$ ./infos.sh qhelloworld-1.0.0-1mdv2009.1.i586.rpm
```

Fabriquer un paquetage

RPM (*Red Hat Package Manager*)

RPM désigne trois choses :

- un programme pour créer ou installer des paquetages,
- un format utilisé dans des paquetages (source ou binaire) créés par le programme rpm,
- un fichier appelé paquetage qui contient les sources ou le binaire ainsi qu'une en-tête d'information sur la méthode d'installation ou de désinstallation du programme.

Du point de vue de l'utilisateur, le programme rpm est un programme de gestion de paquetages. Il pilote en fait toutes les actions sur un paquetage RPM. Il peut ainsi, entre autres :

- installer ou mettre à jour un paquetage en vérifiant ses dépendances,
- exécuter des actions pendant l'installation pour rendre le programme installé prêt à l'emploi,
- restaurer des fichiers d'un paquetage effacés accidentellement,
- dire si un paquetage est déjà installé,
- trouver de quel paquetage provient un fichier particulier,
- vérifier l'installation courante et le respect des dépendances pour tous les paquetages installés,
- ...

Du point de vue du programmeur, le programme rpm est un empaqueteur qui encapsule dans un seul fichier RPM toutes les informations nécessaires à l'installation d'un programme sur une plate-forme donnée.

Il est important de distinguer dès le début les paquetages sources (*.src.rpm*) et les paquetages binaires (*.<archtype>.rpm*).

Les premiers contiennent (*.src.rpm*) l'arborescence entière des sources du programme, plus tout ce que le réalisateur du paquetage a ajouté pour qu'il puisse être configuré, compilé et finalement installé. Cela consiste généralement en un fichier ayant l'extension *.spec* (le fichier utilisé pour dire à rpm les opérations à effectuer pour créer le paquetage binaire) ainsi que des correctifs, si nécessaire.

Les seconds (*.<archtype>.rpm*) contiennent le binaire compilé, ainsi que tous les fichiers (documentation, fichiers de configuration, icones,...) qui seront installés sur le système cible. Ils contiennent aussi la procédure qui installe les fichiers aux emplacements corrects, ainsi que les actions qui rendent le programme opérationnel.

Remarques : bien que RPM ait été conçu à l'origine pour la distribution Red Hat, il a été adopté par d'autres distributions : Mandriva, Suse, etc. La commande rpm est déja installé sur ces systèmes.
Les *rpm* binaires que vous construirez pour Mandriva Linux ne fonctionnent pas forcément sur toutes les distributions, ni même sur toutes les versions de Mandriva, bien que des efforts soient fait pour rester compatible avec Red Hat. Il existe aussi le format deb de la distribution Debian et utilisé aussi par Ubuntu.

Lire : http://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_paquets

 © BTS IRIS 2010 <tv>

Tâches préliminaires

Installation requise

Pour pouvoir construire des RPM, vous devez avoir installé au préalable le paquetage **rpm-build**.

Arborescence

Pour construire des paquetages, RPM a besoin d'une arborescence spéciale dans le dossier personnel de l'utilisateur. Cette arborescence peut être créée en tapant dans une console :

```
$ mkdir -p ~/rpm/{BUILD,RPMS/
{i586,noarch,x86_64},SOURCES,SRPMS,SPECS,tmp}
```

Remarque : Il est dangereux de construire des RPM en tant que root, puisque les fichiers binaires sont installés sur le système avant d'être empaquetés. Il faut donc toujours construire ses RPM en tant qu'utilisateur normal afin de ne jamais polluer accidentellement son système. Pour installer le RPM binaire, il faut être sous root, mais c'est inutile pour construire un *.rpm* ou décompresser un *.src.rpm*.

Vérifiez bien que l'arborescence est de la forme:

- *~/rpm/BUILD* : dossier où se fait la compilation des sources.
- *~/rpm/RPMS* : contient les répertoires, un par architecture, qui contiendront les paquetages binaires générés.
- *~/rpm/RPMS/i586* : le répertoire où seront stockés les paquetages binaires créés pour les processeurs i586.
- *~/rpm/RPMS/x86_64* : le répertoire où seront stockés les paquetages binaires créés pour les processeurs X86_64.
- *~/rpm/RPMS/noarch* : le répertoire où seront stockés les paquetages binaires *noarch* (c'est-à-dire indépendants de l'architecture du processeur). générés. NdT : c'est souvent le cas des applications écrites dans des langages interprétés (php,perl,python,ruby...).
- *~/rpm/SOURCES* : contient les fichiers sources (par exemple monpaquetage.tar.bz2).
- *~/rpm/SPECS*: contient les fameux fichiers *spec* que nous devons écrire.
- *~/rpm/SRPMS* : RPM sources après la construction.
- *~/rpm/tmp* : dossier temporaire de travail pour RPM.

Il faut copier l'archive fournie dans *~/rpm/SOURCES* :

```
$ cp qhelloworld-1.0.0.tar.gz ~/rpm/SOURCES/qhelloworld-1.0.0.tar.gz
```

Configuration pour Mandriva Linux

Pour pouvoir construire des paquetages pour Mandriva Linux, vous devrez au préalable créer le fichier de configuration *.rpmmacros* dans votre répertoire personnel et y copier/coller le contenu ci-dessous :

```
$ vim ~/.rpmmacros
%_topdir                %(echo $HOME)/rpm
%_tmppath               %(echo $HOME)/rpm/tmp

# If you want your packages to be GPG signed automatically, add these three
lines
# replacing 'Mandrivalinux' with your GPG name. You may also use rpm
--resign
# to sign the packages later.
%_signature             gpg
%_gpg_name              Mandrivalinux
%_gpg_path              ~/.gnupg

# Add your name and e-mail into theÂ %packager field below. You may also
want to
# also replace vendor with yourself.
%packager               Thierry Vaira <thierry.vaira@orange.fr>
%distribution           Mandriva Linux
%vendor                 Mandriva

#If you want your packages to have your own distsuffix instead of mdv,add it
#here like this
#%distsuffix             foo
```

Attention, ne pas définir %optflags, puisqu'elle est déjà définie globalement dans le fichier */usr/lib/rpm/rpmrc*.

Le programme rpm est maintenant correctement configuré pour construire des paquetages.

Vérifications préliminaires

Licence

Malgré la prédominance de la licence **GPL**, il existe d'autres licences libres (comme BSD, MIT, etc) ou non libres. Mandriva n'accepte pas de logiciels non libres, sauf dans la section non-free. Elle ne peut pas non plus accepter de logiciels dont la licence ne lui permet pas de les distribuer librement. Vérifiez donc au préalable si un logiciel est acceptable d'après sa licence.

Compression de l'archive

Pour simplifier la maintenance, il est recommandé d'utiliser une méthode de compression. On choisit généralement le .tar.bz2. Évitez de compresser les fichiers au format texte (comme les patchs produits par diff et autres, les fichiers de configuration, les scripts, etc.). Leur compression économiserait très peu d'espace tout en rendant plus difficile la visualisation des changements, au niveau des diffs produits par Subversion.

 © BTS IRIS 2010 <tv>

Le fichier .spec

C'est la partie la plus importante de ce document. Le fichier *spec* contient toutes les informations dont RPM a besoin pour :

1. compiler le programme et construire les RPM binaires et sources,
2. installer ou désinstaller le programme sur la machine de l'utilisateur final.

Le débutant peut être dérouté par le fait que ces deux types d'information sont regroupés dans un seul fichier. Cela est dû à l'arborescence de l'archive tar source, qui contient déjà cette information. Comme la procédure d'installation est extraite au cours du processus d'installation, généralement lancé par un make install dans l'arborescence source, les deux parties sont intimement liées.

En bref, le fichier spec décrit une compilation et une installation simulée, dit à RPM quels fichiers résultants de cette installation doivent être mis dans le paquetage et finalement comment installer ces fichiers dans le système de l'utilisateur. Les commandes sont exécutées en utilisant le shell */bin/sh*, de telle sorte que des commandes comme [-f configure.in] && autoconf sont valides.

Nous n'allons pas exposer ici en détail toutes les possibilités d'un fichier spec. Nous allons nous contenter de passer en revue les options utilisées dans un exemple de fichier spec standard Mandriva.

Pour plus de détails : lire la section 7 (http://www.rpm.org/max-rpm/)

Conseil : plus vous construirez de RPM, plus vous découvrirez d'options. RPM est très extensible. Il est toujours bon d'ouvrir des fichiers *specs* et d'y jeter un coup d'œil pour comprendre leur fonctionnement.

```
$ vim ~/rpm/SPECS/qhelloworld.spec
%define name qhelloworld
%define version 1.0.0
%define release %mkrel 1
Name: %{name}
Summary: Affiche "Hello world !" dans une boite de dialogue
Version: %{version}
Release: %{release}
Source0: %{name}-%{version}.tar.gz
URL: http://www.btsiris.net/~tv/qhellowordl/
Group: Development/KDE and Qt
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-buildroot
License: GPL
#Requires: qt4 => 4.5.2

%description
Affiche "Hello world !" dans une boite de dialogue cree par Qt4

%prep
%setup -q

%build
#configure
qmake
%make
```

```
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/usr/{bin,share/man/man1}
%makeinstall
cp -p $RPM_BUILD_DIR/%{name}-%{version}/usr/bin/%{name}
$RPM_BUILD_ROOT/usr/bin/
cp -p $RPM_BUILD_DIR/%{name}-%{version}/usr/share/man/man1/%{name}.1.*
$RPM_BUILD_ROOT/usr/share/man/man1/

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root)
%doc README NEWS COPYING AUTHORS TODO ChangeLog
%{_mandir}/man1/%{name}.1*
%{_bindir}/%{name}

%changelog
* Mon Nov 15 2010 Thierry Vaira <thierry.vaira@orange.fr> 1.0.0-
1mdv2009.1

- Changes from 0.1.0 to 0.2.0
   * Support de l'installation par make install
   * Ajout de la variable INSTALLS += target extra man dans
qhelloworld.pro
   * Ajout d'une page man (man/man1/qhelloworld.1.lzma)
```

Les explications détaillées d'un fichier .spec sont fournies ci-après pour un autre exemple.

 © BTS IRIS 2010 <tv>

Exemple de contenu d'un fichier .spec

```
%define name    gif2png
%define version 2.0.1
%define release %mkrel 1

Name:           %{name}
Summary:        Tools for converting websites from using GIFs to using PNGs
Version:        %{version}
Release:        %{release}
Source0:        http://www.tuxedo.org/~esr/gif2png/%{name}-%{version}.tar.bz2
Source1:        %{name}-%{version}-mdk-addon.tar.bz2
Patch0:         gif2png-2.0.1-bugfix.patch
URL:            http://www.tuxedo.org/~esr/gif2png/

Group:          Applications/Multimedia
BuildRoot:      %{_tmppath}/%{name}-%{version}-%{release}-buildroot
License:        MIT-like
Requires:       python

%description
Tools for converting GIFs to PNGs. The program gif2png converts GIF files
to PNG files. The Python script web2png converts an entire web tree, also
patching HTML pages to keep IMG SRC references correct.

%prep
%setup -q -a 1
%patch -p1

%build
%configure
%make

%install
rm -rf $RPM_BUILD_ROOT
%makeinstall

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root)
%doc README NEWS COPYING AUTHORS
%{_mandir}/man1/gif2png.1*
%{_mandir}/man1/web2png.1*
%{_bindir}/gif2png
%{_bindir}/web2png

%changelog
* Mon Nov 02 1999 Camille Begnis <camille@mandrakesoft.com> 2.0.1-1mdk
- Upgraded to 2.0.1

* Mon Oct 25 1999 Camille Begnis <camille@mandrakesoft.com> 2.0.0-1mdk
- Specfile adaptations for Mandrake
- add python requirement
- gz to bz2 compression
```

 © BTS IRIS 2010 <tv>

Description du contenu d'un fichier .spec

Un % au début d'une ligne peut signifier soit :

- le début d'une section (prep, build, install, clean)
- un script de macro du shell (setup, patch)
- une instruction utilisée par une section spéciale (defattr, doc, ...)

Vous pouvez voir à quoi correspond une macro à l'aide de l'instruction rpm --eval :

```
$ rpm --eval "%mkrel 1"
1mdv2010.0
```

Section En-tête

```
%define name    gif2png
%define version 2.0.1
%define release %mkrel 1
```

Ces 3 lignes définissent des constantes utilisables par les sections suivantes du spec, constantes qui seront alors nommées *%{name}* , *%{version}* et *%{release}* . *%mkrel* est une macro Mandriva qui doit être utilisée pour ajouter le suffixe « mdv » et la version de la distribution après le numéro de révision du paquet.

Nous pouvons maintenant remplir certains champs d'information pour rpm :

```
Name:           %{name}
```

C'est le nom du paquetage tel qu'il sera identifié sur la machine de l'utilisateur et dans sa base de données de paquetages une fois installé. Noter que "*%{name}* " se réfère au « define » précédent.

```
Version:        %{version}
Release:        %{release}
```

Il est maintenant temps d'expliquer comment est formé le nom d'un paquet. Il est important de toujours respecter ce standard pour que votre travail soit compréhensible par les autres.

Un paquetage binaire se nomme ainsi : *name-version-release.arch.rpm*

- Un paquetage source se nomme ainsi : *name-version-release.src.rpm* (par ex: *gif2png-2.0.1-1mdk.src.rpm* pour notre exemple)

Le nom « name » est généralement celui de l'exécutable principal du paquetage, bien qu'on puisse choisir un autre nom pour des raisons valables.

La « version » est le numéro de version des sources non patchées, numéro qu'on retrouve dans le nom de fichier de l'archive d'origine : *name-version.tar.gz*.

La « release » est un nombre, incrémenté à chaque nouvelle construction du paquetage, suivi d'un suffixe indiquant pour quelle distribution a été construit le paquetage, par exemple "mdv2008.1" pour "Mandriva 2008.1". Notez que cette extension est obligatoire. Un paquetage peut être reconstruit pour de multiples raisons : l'ajout d'un nouveau correctif (patch) aux sources, une modification du fichier spec, l'ajout d'une icone, etc.

 © BTS IRIS 2010 <tv>

Summary: tools for converting websites from using GIFs to using PNGs

Le « Summary » est une courte description du paquetage, en une seule ligne. Il est important qu'on puisse comprendre ce que fait le paquet à partir de ce champ, et que cela ne dépasse pas une certaine limite (80 caractéres).

Source0: http://www.tuxedo.org/~esr/gif2png/%{name}-%{version}.tar.bz2

Cette ligne indique à rpm le fichier source à utiliser pour construire ce paquetage. A noter que le nom est précédé d'une URL complète (optionnelle) pointant vers le site où sont disponibles les sources d'origine ; rpm enlève l'URL, ne gardant que le nom de fichier et cherche ce fichier dans le dossier *~/rpm/SOURCES*. Bien que l'URL complète soit optionnelle, elle est recommandée pour que les utilisateurs sachent où trouver les nouvelles sources pour les mettre à jour et recompiler le programme. De plus, cela permet à des outils comme mdvsys de reconstruire automatiquement de nouvelles versions S'il y a plus d'un fichier source, on ajoute d'autres lignes sources, comme : Source1: ..., puis Source2: ..., etc.

Patch0: gif2png-2.0.1-bugfix.patch

Les fichiers patchs doivent être placés dans le dossier même dossier que les sources (~/rpm/SOURCES). Comme pour les sources, il peut y avoir plusieurs correctifs. Ils seront alors désignés dans le fichier spec par des lignes Patch1: ..., puis Patch2: ..., etc.

URL: http://www.tuxedo.org/~esr/gif2png/

Cette ligne (optionnelle mais recommandée) pointe vers la page Web du programme.

Group: Multimedia

Ceci indique à rpm à quel emplacement de l'arborescence générale des paquetages placer ce paquetage. Cette information est utilisée par les gestionnaires de paquets tels que rpmdrake.

BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-buildroot

Cette ligne très importante ne peut être omise. Elle demande à RPM, pour installer le programme, d'utiliser un dossier racine spécial (un pseudo "/") sur la machine où se fait la compilation. Deux raisons à cela :

1. Quand on construit un RPM, on n'a pas d'accès root à la machine et on ne peut donc pas installer le paquetage dans les dossiers systèmes habituels.
2. Si on fait l'installation dans l'arborescence système d'une machine, les fichiers du paquetage vont se mélanger avec les autres et, surtout, cela pourrait même être dangereux si le paquetage est déja installé. Beaucoup de gens utilisent */var/tmp* ou */tmp* comme « BuildRoot ». Ça ne pose pas nécessairement de problème si on est seul utilisateur de la machine, mais s'il y a plusieurs utilisateurs sur cette machine et qu'ils compilent le même paquetage au même moment, rpm va planter. C'est pourquoi il est préférable de définir *%{_tmppath}* vers un sous-dossier de votre répertoire personnel.

```
License:        MIT-like
```

Ce champ (qui remplace « Copyright ») définit la licence choisie par le propriétaire du Copyright (ou droits d'auteur) qui s'applique au programme empaqueté. C'est le plus souvent la license GPL.

```
Requires:       python
```

Cette ligne a été ajoutée parce qu'un des programmes du paquetage est un script python. Il a donc besoin de l'interpréteur python pour fonctionner. On peut optionnellement demander une numéro de version minimal en ajoutant un signe supérieur (ou égal), par exemple : Requires: python >= 1.5.1. En pratique, les nouvelles versions de rpm arrive à rajouter la dépendance automatiquement.

```
%description
Tools for converting GIFs to PNGs. The program gif2png converts GIF files
to PNG files. The Python script web2png converts an entire web tree, also
patching HTML pages to keep IMG SRC references correct.
```

Ce champ est spécial parmi ceux de l'en-tête du fichier spec, car c'est un texte entier pouvant faire plusieurs lignes ou paragraphes si nécessaire. C'est une description complète du programme qui va être installé, pour aider l'utilisateur à décider s'il veut installer ou non ce paquetage.
En fait, pour améliorer la lisibilité des fichiers spec, les traductions des champs « Summary » et « description » sont stockés dans un fichier spécial nommé *<package>.po*. Cette méthode suppose que tous les textes d'un fichier .spec sont écrits en anglais.

Section Prep

```
%prep
%setup -q -a 1
%patch0 -p1
```

Cette section contient le script exécuté en premier par rpm. Son rôle est de :

- créer le dossier racine pour la construction (dans BUILD),
- décompresser les sources originales dans le dossier *~/rpm/BUILD*,
- appliquer aux sources les correctifs éventuels.

Elle peut être suivi de toute commande voulue par le créateur du paquetage pour que les sources soient prêts pour la compilation.

```
%setup -q -a 1
```

Ceci est un script pré-défini qui :

- exécute une commande cd dans l'arbre de compilation,
- extrait les sources (silencieusement, -q),
- change le propriétaire et les permissions des fichiers sources.

Par défaut, il n'extrait que le premier source. Il faut utiliser des paramètres pour des sources supplémentaires: dans notre exemple, -a 1 dit que nous voulons aussi extraire le source numéro 1.

La macro *%setup* a d'autres options intéressantes:

- -c name: l'option « -c » demande de créer d'abord le répertoire racine "nom", puis de s'y déplacer par cd et de décompresser Source0. C'est utile pour certains paquetages qui ont été "tar.bz-ippés" sans répertoire parent.
- -D : ne pas effacer le répertoire avant décompression. Cela ne sert que s'il y a plus d'une macro %setup. A utiliser seulement dans les %setup qui suivent le premier (mais jamais dans le premier).
- -T : cette option remplace l'action par défaut de décompresser la Source (et nécessite alors un -b 0 ou -a 0 pour décompresser le fichier source principal). Nécessaire quand il y a des sources secondaires.
- -n <name> : à utiliser si le nom du rpm est différent de celui en lequel se décompresse le Source. Par exemple, si le rpm se nomme program-version-revision et que le Source se décompresse en program-version-date, le processus de build du rpm ne pourra pas entrer (cd) dans le répertoire program-version, il faut alors utiliser « -n program-version-date », pour que rpm sache dans quel nouveau répertoire il doit continuer.

`%patch0 -p1`

Cette macro applique le patch aux sources ; son paramètre "-p<numero>" est passé au programme patch.

Section Build

`%build`

Cette section contient le script qui compile réellement le logiciel. Il se compose de commandes lancées sur l'arborescence des sources décompressés.

`%configure`

C'est la ligne, qui configure les sources utilisant autoconf. *%configure* lance un ./configure avec de nombreuses options telles que export CFLAGS="$RPM_OPT_FLAGS" avant le configure, et des options telles que i586-mandriva-linux-gnu --prefix=/usr –datadir=/usr/share etc.
Ces arguments ne sont pas toujours gérés par le script de configuration. Dans ce cas, il faut en découvrir la raison puis lancer ./configure avec les paramètres appropriés.

`%make`

C'est une simple macro qui, de base, exécute un make.

Section Install

`%install`

Cette section contient le script qui installe vraiment le paquet dans le dossier d'installation simulée : *$RPM_BUILD_ROOT*.

Ce sont les commandes qui rendent le logiciel fonctionnel sur le système de l'utilisateur.

`rm -rf $RPM_BUILD_ROOT`

C'est la première commande exécutée dans la section *%install* qui nettoie le dossier d'une éventuelle précédente installation.

```
%makeinstall
```

Cette ligne installe le logiciel dans le dossier d'installation simulée pour des sources préparées par autoconf. Cette macro se développe en "make install" avec diverses options pour que le logiciel soit installé dans le dossier d'installation simulée *$RPM_BUILD_ROOT*, par exemple
prefix=$RPM_BUILD_ROOT%{_prefix} bindir=$RPM_BUILD_ROOT%{_bindir}
etc. Il arrive que le script de configuration soit partiellement cassé auquel cas il faudra aller fouiner dans les fichiers *Makefile* pour deviner les paramètres optionnels à passer pour que le logiciel s'installe correctement. Une des situations les plus courantes est d'avoir à utiliser
make DESTDIR=$RPM_BUILD_ROOT install

Pour économiser à la fois de l'espace disque et du temps de téléchargement, Mandriva utilise lzma (anciennement bzip2) pour compresser les pages man et info. Cet aspect est cependant pris en charge en standard par la version de rpm de Mandriva.

```
%clean
```

Cette section nettoie le dossier de construction *$RPM_BUILD_ROOT*.

```
rm -rf $RPM_BUILD_ROOT
```

C'est ici que se fait le travail.

Section Files

```
%files
```

Cette section est une liste de fichiers à prendre dans le dossier d'installation simulée pour les incorporer au paquetage. Voir le manuel pour d'autres options absentes de cet exemple simple.
La liste des fichiers doit être écrite à la main dans le fichier spec. On peut la construire en listant tous les fichiers créés par rpm dans le répertoire de construction. Pour cela, on exécute un rpm -bi mypackage.spec pour arrêter le processus de construction juste après l'installation simulée. On examine alors le contenu du dossier d'installation simulée, *~/rpm/tmp/gif2png-buildroot* dans notre cas, pour voir quels fichiers on veut mettre dans le paquetage (le plus souvent, on les met tous).

Note sur la structure des répertoires : les fichiers installés par votre paquetage doivent suivre les recommandations FHS disponibles sur http://www.pathname.com/fhs

```
%defattr(-,root,root)
```

Ce champ définit les attributs à appliquer à chaque fichier qui sera copié sur le système de l'utilisateur.
Les quatres arguments donnés signifient:

- - : tous les attributs des fichiers réguliers restent inchangés,
- root : le propriétaire du fichier est root,
- root : le groupe propriétaire du fichier est root,
- (optionnel) 0755 : les attributs du groupe propriétaire appliqués à tous les répertoires possédés par ce paquetage sont 0755 (rwxr-xr-x).

```
%doc README NEWS COPYING AUTHORS
```

Le champ spécial *%doc* désigne les fichiers qui font partie de la documentation du paquetage. Les dits fichiers seront placés dans */usr/share/doc/gif2png/*. Ce dossier sera aussi automatiquement créé. Les fichiers spécifiés par *%doc* sont placés relativement au répertoire source décompressé dans *BUILD*.

```
%{_mandir}/man1/gif2png.1*
%{_mandir}/man1/web2png.1*
```

Il est recommandé de lister ici aussi chaque fichier man ou info séparément.
Aussi, vous vous demandez peut être pourquoi dire gif2png.1* et non pas gif2png.1.lzma. C'est pour préserver la compatibilité avec les autres systèmes qui pourraient utiliser la compression gzip. Si on trouve de telles références à lzma dans des fichiers specs, les remplacer par un joker. Le plus souvent, on peut aussi utiliser %{_mandir}/man1/* qui prendra tous les fichiers qu'il trouvera.

```
%{_bindir}/gif2png
%{_bindir}/web2png
```

On peut voir qu'il y a des macros pour chaque type de chemin nécessaire. Voici les plus utiles (regardez dans */usr/lib/rpm/macros* pour les avoir toutes) : *%{_prefix}* , *%{_bindir}* , *%{_sbindir}* , *%{_datadir}* , *%{_libdir}* , *%{_sysconfdir}* , *%{_mandir}* , *%{_infodir}* . Pour des jeux, utilisez *%{_gamesbindir}* et *%{_gamesdatadir}*

Section Changelog

```
%changelog
```

Cette section garde une trace des changements apportés au paquetage. Chaque paragraphe de cette section correspond à une nouvelle version du paquetage avec augmentation du numéro de « release » du paquetage (voir du numéro de version du paquetage si il empaquete une nouvelle version du logiciel). Ces paragraphes doivent respecter la structure suivante :

```
* Mon Nov 02 1999 Camille Begnis <camille@mandrakesoft.com> 2.0.1-1mdk
```

- La première ligne du paragraphe commence par * avec, dans l'ordre et séparés par un espace :
- 3 lettres pour le jour de la semaine (en anglais)
- 3 lettres pour le mois (en anglais)
- 2 chiffres pour jour du mois (en anglais)
- 4 chiffres pour l'année
- Prénom du créateur du paquetage
- Nom du créateur du paquetage
- e-mail du créateur du paquetage entre <>.
- version et release des modifications.

```
- Upgraded to 2.0.1
```

Ensuite, une ligne, commençant par un -, par modification appliquée au paquetage.

La construction

Notre fichier *spec* est enfin complet. Prenez une grande inspiration, asseyez vous et tapez :

```
$ rpmbuild -ba qhelloworld.spec
```

La commande rpmbuild a créé trois paquetages :

- **~/rpm/rpm/RPMS/i586/qhelloworld-1.0.0-1mdv2009.1.i586.rpm**
- ~/rpm/rpm/RPMS/i586/qhelloworld-debug-1.0.0-1mdv2009.1.i586.rpm
- ~/rpm/rpm/SRPMS/qhelloworld-1.0.0-1mdv2009.1.src.rpm

On peut ajouter l'option --clean qui nettoie le répertoire *BUILD* une fois le paquetage construit.

Vous allez voir apparaître toutes les commandes qu'il effectue pendant la construction. La toute dernière ligne donne le résultat :

- + exit 0 = La construction a réussie.
- Si vous n'avez pas le message ci-dessus, la construction a sans doute échouée.

Si vous êtes dans le second cas, regardez les options de construction de RPM (man rpm) pour déboguer votre travail, regardez les fichiers *spec* d'autres personnes, etc...

Il y a une manière très propre de construire un paquetage : utilisez rpm -bs --rmspec --rmsource, pour supprimer tout ce qui provient de la construction d'origine, puis faites un rpm --rebuild.
scripts de pré- et post-installation

Un paquetage RPM est en fait bien plus qu'une simple archive contenant des fichiers à décompresser dans des répertoires spécifiques sur le système client.

Le système fournit au programmeur une possibilité intéressante : les scripts de pré- et post-installation. Grâce à eux, le créateur du paquetage peut écrire un morceau de code qui sera exécuté sur la machine cliente pendant l'installation ou la suppression du paquetage. Il faut connaître certaines particularités de ces scripts pour en tenir compte : premièrement, ils doivent tenir dans un tampon de 8192 octets, deuxièmement, ils ne doivent pas être interactifs. Toute interaction avec l'utilisateur est à proscrire, puisqu'elle empêcherait les procédures automatiques d'installation de RPM de fonctionner.

Ces scripts se composent de n'importe quel ensemble de commandes sh valides. En voici quatre :

- *%pre* : ce script s'exécute juste avant l'installation du paquetage sur le système.
- *%post* : ce script s'exécute juste après l'installation du paquetage sur le système.
- *%preun* : ce script s'exécute juste avant la désinstallation du paquetage du système.
- *%postun* : ce script s'exécute juste après la désinstallation du paquetage du système.

Ces scripts ont un très grand rayon d'action et il faut les mettre au point avec le plus grand soin pour ne pas perturber le système hôte. Ne pas oublier que ces scripts s'exécuteront en tant que root... Ce sont les tâches système qu'un administrateur système accomplirait pour installer un nouveau programme sur le système. Par exemple :

- Ajouter un job cron qui lance le programme à intervalles fixes.
- Lancer chkconfig pour lancer le démon au moment du boot.
- ...

Installer le paquetage créé

Pour installer le paquetage (sous **root**) :

```
# rpm -ivh ./qhelloworld-1.0.0-1mdv2009.1.i586.rpm
Préparation...              ################################# [100%]
   1:qhelloworld            ################################# [100%]
```

On peut tester :

```
$ qhelloworld

$ man qhelloworld
```

Puis, pour désinstaller le paquetage (sous **root**) :

```
# rpm -ev qhelloworld-1.0.0-1mdv2009.1.i586
```

Et bien évidemment, cela ne marche plus :

```
$ qhelloworld
bash: /usr/bin/qhelloworld: Aucun fichier ou dossier de ce type

$ man qhelloworld
Il n'y a pas de page de manuel pour qhelloworld.
```

Ajout de l'application dans les menus KDE

Pour construire des RPM pour Mandriva Linux, d'autres macros permettent de simplifier le fichier *spec*. Le menu système. Depuis Mandriva Linux 2007.0, les menus XDG sont utilisés, remplaçant les anciens menus Debian.

```
%post
%{update_menus}

%postun
%{clean_menus}
```

On va donc modifier (parties en gras) le fichier **`qhelloworld.spec`** afin d'y intégrer l'ajout de l'application dans les menus de KDE :

```
%define name qhelloworld
%define version 1.0.0
%define release %mkrel 1
Name: %{name}
Summary: Affiche "Hello world !" dans une boite de dialogue
Version: %{version}
Release: %{release}
Source0: %{name}-%{version}.tar.gz
URL: http://www.btsiris.net/~tv/qhellowordl/
Group: Development/KDE and Qt
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-buildroot
```

```
BuildRequires: desktop-file-utils
License: GPL
#Requires: qt4 => 4.5.2

%description
Affiche "Hello world !" dans une boite de dialogue creee par Qt4

%prep
%setup -q

%build
#configure
qmake
%make

%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/usr/{bin,share/man/man1}
%makeinstall
cp -p $RPM_BUILD_DIR/%{name}-%{version}/usr/bin/%{name} $RPM_BUILD_ROOT/usr/bin/
cp -p $RPM_BUILD_DIR/%{name}-%{version}/usr/share/man/man1/%{name}.1.*
$RPM_BUILD_ROOT/usr/share/man/man1/
mkdir -p %{buildroot}%{_datadir}/applications
cat > %{buildroot}%{_datadir}/applications/mandriva-%{name}.desktop <<EOF
[Desktop Entry]
Encoding=UTF-8
Name=QHelloWorld
GenericName=Qt Application
Comment=Affiche Hello world
Exec=%{_bindir}/%{name}
Icon=qdevelop
Terminal=false
Type=Application
StartupNotify=true
MimeType=application/x-qdevelop;
Categories=Qt;Development;IDE;
EOF

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root)
%doc README NEWS COPYING AUTHORS TODO ChangeLog
%{_mandir}/man1/%{name}.1*
%{_bindir}/%{name}
%{_datadir}/applications/mandriva-%{name}.desktop

%changelog
* Mon Nov 15 2010 Thierry Vaira <thierry.vaira@orange.fr> 1.0.0-1mdv2009.1

- Changes from 0.1.0 to 0.2.0
   * Support de l'installation par make install
   * Ajout de la variable INSTALLS += target extra man dans qhelloworld.pro
   * Ajout d'une page man (man/man1/qhelloworld.1.lzma)

%post
%{update_menus}

%postun
%{clean_menus}
```

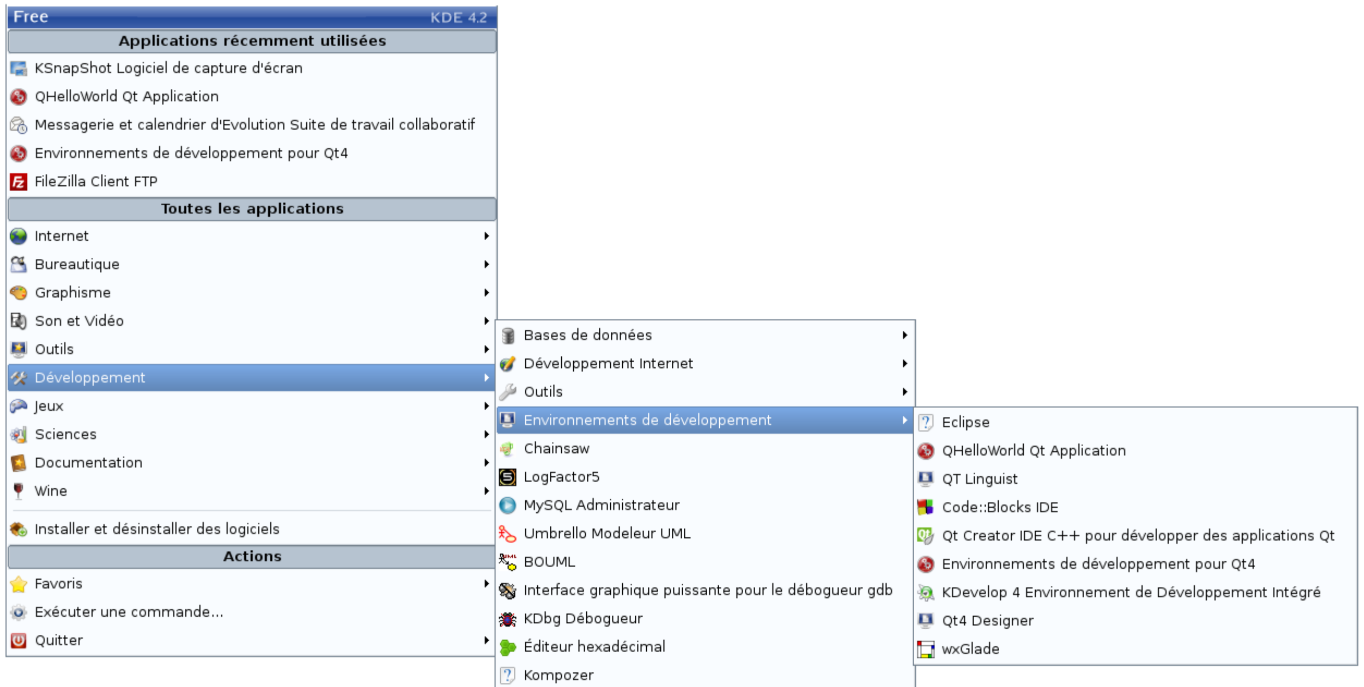
Il faut bien évidfemment reconstruire le paquetage :

```
$ rpmbuild -ba qhelloworld.spec
```

Puis, ré-installer le paquetage (sous **root**) :

```
# rpm -ivh ./qhelloworld-1.0.0-1mdv2009.1.i586.rpm
Préparation...              ################################# [100%]
   1:qhelloworld            ################################# [100%]
```

Une fois le paquetage installé, on obtient :



Tests

Avant d'installer le paquetage, on va récupérer des informations sur celui-ci.

Pour cela, on va créer un petit script :

```
$ vim infosrpms.sh
#!/bin/bash
if test -e "$1"
then
   echo
   echo Informations sur le paquetage "$1" :
   rpm -qpi $1
   echo
   echo Liste des fichiers :
   rpm -qpl $1
   echo
   echo Dependances :
   rpm -qpR $1
   echo
   echo Fichiers de documentation :
   rpm -qpd $1
   echo
   echo "Changelog :"
   rpm -qp --changelog $1
   echo
else
   echo "Paquetage ""$1"" introuvable ou manquant !"
   echo
fi
```

On exécute le script sur notre paquetage :

```
$ ./infos.sh qhelloworld-1.0.0-1mdv2009.1.i586.rpm

Informations sur le paquetage qhelloworld-1.0.0-1mdv2009.1.i586.rpm :
Name        : qhelloworld                  Relocations: (not relocatable)
Version     : 1.0.0                        Vendor: Mandriva
Release     : 1mdv2009.1                   Build Date: ven. 16 avril 2010 11:11:49
CEST
Install Date: (not installed)              Build Host: localhost
Group       : Development/KDE and Qt       Source RPM: qhelloworld-1.0.0-
1mdv2009.1.src.rpm
Size        : 32873                        License: GPL
Signature   : (none)
Packager    : Thierry Vaira <thierry.vaira@orange.fr>
URL         : http://www.btsiris.net/~tv/qhellowordl/
Summary     : Affiche "Hello world !" dans une boÃ®te de dialogue
Description :
Affiche "Hello world !" dans une boîte de dialogue créée par Qt4

Liste des fichiers :
/usr/bin/qhelloworld
/usr/share/doc/qhelloworld
/usr/share/doc/qhelloworld/AUTHORS
/usr/share/doc/qhelloworld/COPYING
/usr/share/doc/qhelloworld/ChangeLog
/usr/share/doc/qhelloworld/NEWS
```

```
/usr/share/doc/qhelloworld/README
/usr/share/doc/qhelloworld/TODO
/usr/share/man/man1/qhelloworld.1.lzma

Dependances :
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
rpmlib(CompressedFileNames) <= 3.0.4-1
libc.so.6
libc.so.6(GLIBC_2.0)
libgcc_s.so.1
libgcc_s.so.1(GCC_3.0)
libQtCore.so.4
libQtGui.so.4
libstdc++.so.6
libstdc++.so.6(GLIBCXX_3.4)
rtld(GNU_HASH)
rpmlib(PayloadIsLzma) <= 4.4.6-1

Fichiers de documentation :
/usr/share/doc/qhelloworld/AUTHORS
/usr/share/doc/qhelloworld/COPYING
/usr/share/doc/qhelloworld/ChangeLog
/usr/share/doc/qhelloworld/NEWS
/usr/share/doc/qhelloworld/README
/usr/share/doc/qhelloworld/TODO
/usr/share/man/man1/qhelloworld.1.lzma

Changelog :
* lun. nov. 15 2010 Thierry Vaira <thierry.vaira@orange.fr> 1.0.0-1mdv2009.1
- Changes from 0.1.0 to 0.2.0
   * Support de l'installation par make install
   * Ajout de la variable INSTALLS += target extra man dans qhelloworld.pro
   * Ajout d'une page man (man/man1/qhelloworld.1.lzma)
```

Une alternative : checkinstall

Une autre façon vraiment simple de construire des RPM pour une utilisation personnelle est d'installer le paquet checkinstall ; de compiler la source comme d'habitude (./configure && make && sudo make install), mais en remplaçant make install par checkinstall. Ceci automatise la construction du RPM, tout en étant vraiment simple d'utilisation. L'avantage consiste dans le fait que vous ne contournez pas le manager RPM en compilant à partir des sources. (Cependant, si vous souhaitez distribuer vos RPM, il est fortement conseillé d'utiliser la méthode vue précédemment.)

Annexe 1 : Qt

Qt est une bibliothèque logicielle orientée objet et développée en C++ par Qt Development Frameworks, filiale de Nokia. Elle offre des composants d'interface graphique (*widgets*), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc.

Qt permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source. Les environnements supportés sont les Unix (dont Linux) qui utilisent le système graphique X Window System, Windows et Mac OS X.

Qt est notamment connu pour être la bibliothèque sur laquelle repose l'environnement graphique KDE, l'un des environnements de bureau les plus utilisés dans le monde Linux.

Historique

Quasar Technologies est créé le 4 mars 1994 et renommé six mois plus tard en Troll Tech, puis Trolltech, puis Qt Software et enfin Qt Development Frameworks. Le 28 janvier 2008, Nokia lance une OPA amicale pour racheter Qt et Trolltech. Trolltech, renommé en Qt Software, devient une division de Nokia.

Il existe une version pour les systèmes embarqués, Qt/Embedded, connue depuis sous le nom de Qtopia, et publiée pour la première fois en 2000.

Depuis, Qt4 sépare la bibliothèque en modules :

- QtCore : pour les fonctionnalités non graphiques utilisées par les autres modules ;
- QtGui : pour les composants graphiques ;
- QtNetwork : pour la programmation réseau ;
- QtOpenGL : pour l'utilisation d'OpenGL ;
- QtSql : pour l'utilisation de base de données SQL ;
- QtXml : pour la manipulation et la génération de fichiers XML ;
- QtDesigner : pour étendre les fonctionnalités de Qt Designer, l'assistant de création d'interfaces graphiques ;
- QtAssistant : pour l'utilisation de l'aide de Qt [6];
- Qt3Support : pour assurer la compatibilité avec Qt 3.
- et de nombreux autres modules, etc.

Licences

Le projet d'environnement graphique KDE a dès le début utilisé la bibliothèque Qt. Mais avec le succès de cet environnement, une certaine partie de la communauté du logiciel libre a critiqué la licence de Qt qui était propriétaire et incompatible avec la GNU GPL utilisée par KDE. Ce problème fut résolu par la société Trolltech qui mit la version Unix/Linux de Qt sous licence GNU GPL lorsque l'application développée était également sous GNU GPL. Pour le reste, c'est la licence commerciale qui entre en application. Cette politique de double licence a été appliquée uniquement pour Unix dans un premier temps, mais depuis la version 4.0 de Qt, elle est appliquée pour tous les systèmes.
Le 14 janvier 2009, Trolltech annonce qu'à partir de Qt 4.5, Qt sera également disponible sous licence LGPL v2.1[12]. Cette nouvelle licence permet ainsi des développements de logiciels propriétaires, sans nécessiter l'achat d'une licence commerciale auprès de Qt Development Frameworks. Ce changement, voulu par Nokia pour faire en sorte que Qt soit utilisé par un maximum de projets, est rendu possible par le fait que Nokia peut se passer des ventes des licences commerciales, contrairement à Trolltech qui ne pouvait pas se passer de cette source de revenus.

Qt Designer est un logiciel qui permet de créer des interfaces graphiques Qt dans un environnement convivial. L'utilisateur, par glisser-déposer, place les composants d'interface graphique et y règle leurs propriétés facilement. Les fichiers d'interface graphique sont formatés en XML et portent l'extension .ui. Lors de la compilation, un fichier d'interface graphique est converti en classe C++ par l'utilitaire uic.

Environnement de développement intégré (EDI ou IDE)
Qt Creator est l'environnement de développement intégré dédié à Qt et facilite la gestion d'un projet Qt. Son éditeur de texte offre les principales fonctions que sont la coloration syntaxique, le complètement, l'indentation, etc... Qt Creator intègre en son sein les outils Qt Designer et Qt Assistant ;
Même si Qt Creator est présenté comme l'environnement de développement de référence pour Qt, il existe des modules Qt pour les environnements de développement Eclipse et Visual Studio. Il existe d'autres EDI dédiés à Qt et développés indépendamment de Nokia, comme QDevelop et Monkey Studio.

QDevelop est un environnement de développement intégré libre pour Qt. Le but de QDevelop est de fournir dans les environnements les plus utilisés, Linux, Windows et Mac OS X d'un outil permettant de développer en Qt de la même manière avec un IDE unique. Il intègre également les outils Qt-Designer pour la création d'interface graphique et Qt-Linguist pour le support de l'internationalisation.

KDevelop est un environnement de développement intégré (IDE) pour KDE. Il intègre également les outils Qt-Designer pour la création d'interface graphique et Qt-Linguist pour la gestion de l'internationalisation.

Autres bibliothèques généralistes multi-plateformes
Parmi les plus connus :

- **GTK+**, utilisée par l'environnement graphique GNOME
- **wxWidgets** (anciennement wxWindows)

Conclusion
De plus en plus de développeurs utilisent Qt, y compris parmi de grandes entreprises. On peut notamment citer : Google, Adobe Systems, Asus, Samsung, Philips, ou encore la NASA et bien évidemment Nokia.

Annexe 2 : Qmake

Qt se voulant un environnement de développement portable et ayant le MOC comme étape intermédiaire avant la phase de compilation/édition de liens, il a été nécessaire de concevoir un moteur de production spécifique. C'est ainsi qu'est conçu le programme qmake.

Ce dernier prend en entrée un fichier (avec l'extension .pro) décrivant le projet (liste des fichiers sources, dépendances, paramètres passés au compilateur, etc...) et génère un fichier de projet spécifique à la plateforme. Ainsi, sous les systèmes UNIX qmake produit un Makefile qui contient la liste des commandes à exécuter pour génération d'un exécutable, à l'exception des étapes spécifiques à Qt (génération des classes C++ lors de la conception d'interface graphique avec Qt Designer, génération du code C++ pour lier les signaux et les slots, ajout d'un fichier au projet, etc.).

Le fichier de projet est fait pour être très facilement éditable par un développeur. Il consiste en une série d'affectations de variables.

Manuel : http://doc.trolltech.com/4.3/qmake-manual.html

 © BTS IRIS 2010 <tv>