

Ma première application Android avec Qt : partie 1

Découvrez comment créer un jeu mobile avec QML et JavaScript

Table des matières

- I. De Cordova à Qt, le JavaScript pour tout faire
- II. Parenthèse sur Qt
- III. Installation avec Ubuntu
- IV. Création d'un projet Qt Quick
- V. Découverte de Qt Creator, Qt Quick et JavaScript
 - V-A. Interface de Qt Creator
 - V-B. Ajout du fichier JavaScript
 - V-C. Inclure le fichier JavaScript dans le fichier QML
 - V-D. Coder de la logique également au sein du QML avec les fonctions
 - V-E. Invoquer un objet QML en JavaScript
- VI. Conclusion
- VII. Remerciements


Développer une application mobile avec une courbe d'apprentissage très courte.

Première partie : l'outil, l'EDI et les librairies

7 commentaires ★★★★★

Article lu 13487 fois.

L'auteur

Michael Bertocchi 

L'article

Publié le 26 février 2016

Version PDF Version hors-ligne

ePub, Azw et Mobi

Liens sociaux



I. De Cordova à Qt, le JavaScript pour tout faire ▲

Suite à mon précédent article sur l'utilisation de Cordova pour écrire une application Android, j'ai souhaité continuer mon portage de jeux web sur Android via cette bibliothèque.

Malheureusement, je me suis heurté à un problème de performance pour mon application « ShootThemUp ». Son écriture terminée, j'ai souhaité la tester sur deux tablettes différentes exécutant deux versions d'Android. Là, je fus étonné de voir une telle différence de fluidité pour un même jeu :

- sur mon ordinateur au sein du navigateur Chrome ;
- sur ma tablette en tant qu'application « Cordova » ;
- sur la même tablette via le navigateur Chrome.

En cherchant un peu, j'ai appris que le moteur d'interprétation web qu'utilise Cordova pour empaqueter les applications n'est pas le même que celui du navigateur internet d'Android.

Ces différences sont encore accentuées par le fait que, entre les deux versions d'Android, Google a changé ce moteur par celui de Chrome, avec à la clef une meilleure performance, mais encore loin de celle du navigateur interne...

Bref, il me fallait une autre solution plus performante, mais avec une courbe d'apprentissage aussi légère que possible pour porter cette application.

Pour améliorer la lecture, cet article est divisé en deux. Ici, nous nous concentrerons sur notre boîte à outils, puis, dans la seconde partie, nous verrons comment créer un petit jeu type « shoot them up » : des ennemis tomberont du haut de l'écran et le joueur pourra les éliminer en cliquant dessus avec la souris/doigt.

II. Parenthèse sur Qt ▲

Qt est soutenu par la Qt Company et permet de faire du développement multiplateforme (framework et éditeur disponible sur Windows/Linux/Mac) et également avec une cible de génération multiplateforme (Windows, Linux, OS X, iOS, Android...)

Ce qui nous intéresse, ici, c'est sa capacité à générer des applications Android, avec une courbe d'apprentissage légère. Bien que, par défaut, Qt propose de développer en C++, nous allons voir que l'on peut développer en QML/JavaScript.

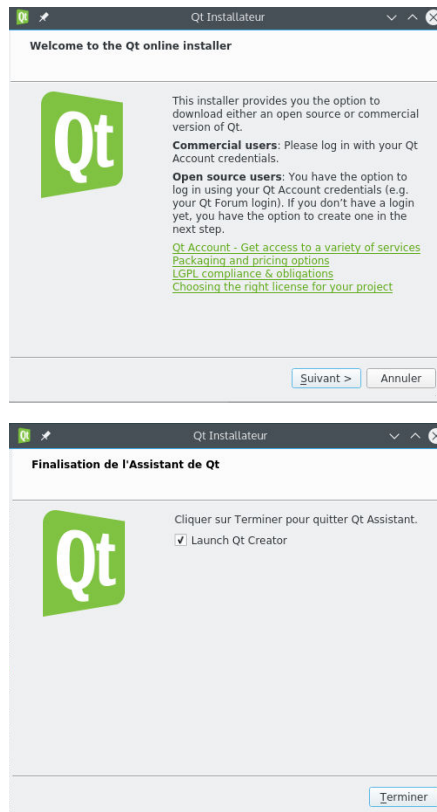
Oui, vous avez bien lu : nous allons ici faire un modeste jeu en JavaScript, appuyé par du QML pour la partie graphique.

Vous verrez d'ailleurs que le QML (qui ressemble à du CSS ou du YAML, pour ceux qui connaissent) ne se contentera pas de l'affichage, c'est en effet un langage bien plus intéressant et puissant.

III. Installation avec Ubuntu ▲

Développant sous GNU/Linux, les instructions d'installation correspondent à Ubuntu (et dérivés).

Vous avez ici deux possibilités : soit passer par votre gestionnaire de paquets habituel (en mode graphique ou non) et installer les paquets qtcreator (les dépendances suivront), soit vous rendre à l'adresse de téléchargement de la version open source.



L'installation terminée, vous avez d'une part votre framework Qt et de l'autre l'EDI proposé et conseillé : Qt Creator.

Installez maintenant le SDK Android (qui sera nécessaire pour générer l'application finale). Rendez-vous sur le site officiel pour le télécharger. Dézippez l'archive téléchargée dans un répertoire, par exemple androidSdk, dans votre répertoire home, ce qui donnera /home/mika/androidSdk/adt-bundle-linux-x86_64-20140702.

Modifiez ensuite la variable d'environnement PATH pour achever l'installation. Dans votre fichier .bashrc, ajoutez les lignes suivantes :

Sélectionnez

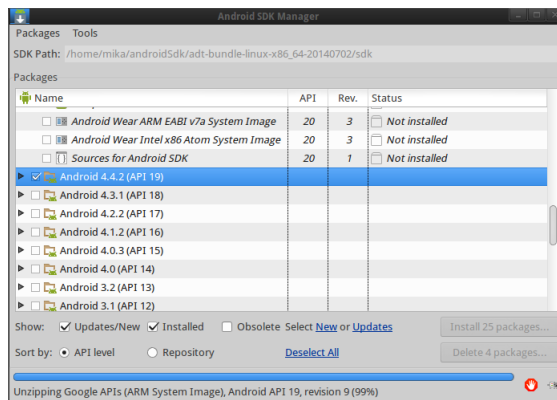
```
export ANDROID_HOME="/home/mika/androidSdk/adt-bundle-linux-x86_64-20140702/sdk"
export PATH="$ANDROID_HOME/tools:$PATH"
export PATH="$ANDROID_HOME/platform-tools:$PATH"
```

Ces lignes définissent une variable d'environnement indiquant le répertoire contenant le SDK Android et ajoutent au PATH les outils d'Android.

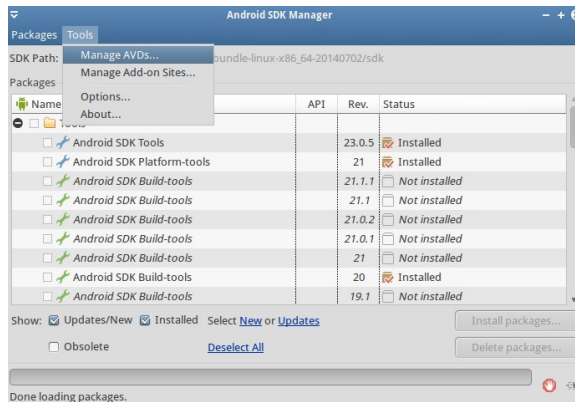
Ajoutez maintenant le SDK pour la version qui sera utilisée (4.4.2). Dans un terminal, lancez :

Sélectionnez

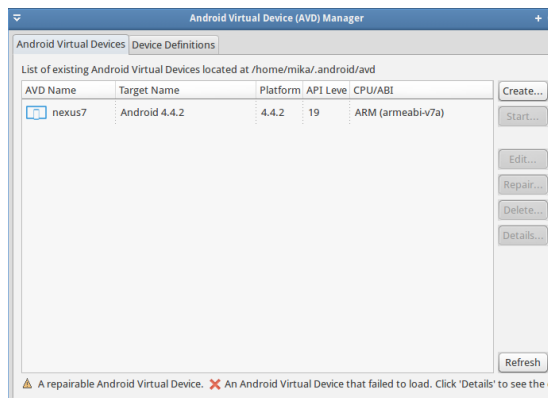
```
android
```



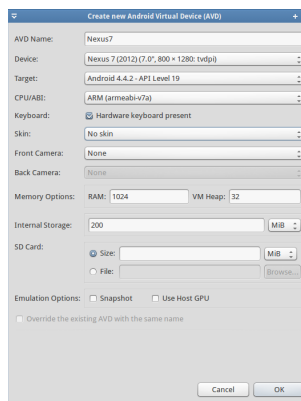
Lancez SDK Manager pour installer API 19 (Android 4.4.2) :



Cliquez sur « create » pour créer un nouvel émulateur :



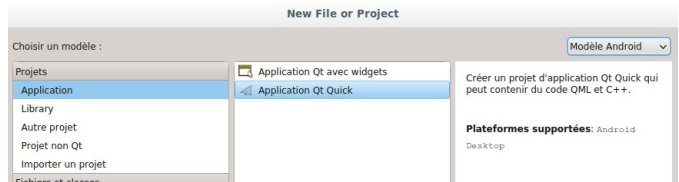
Enfin installez un émulateur :



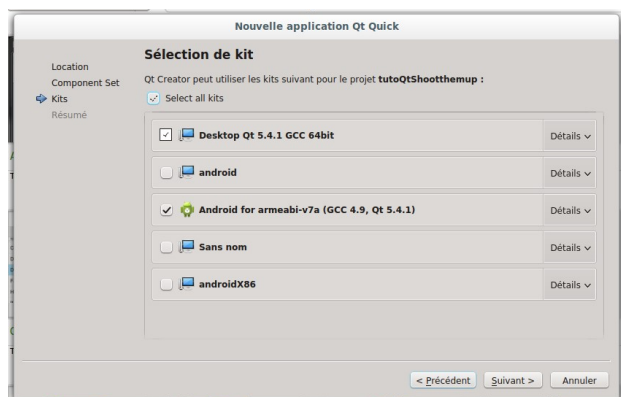
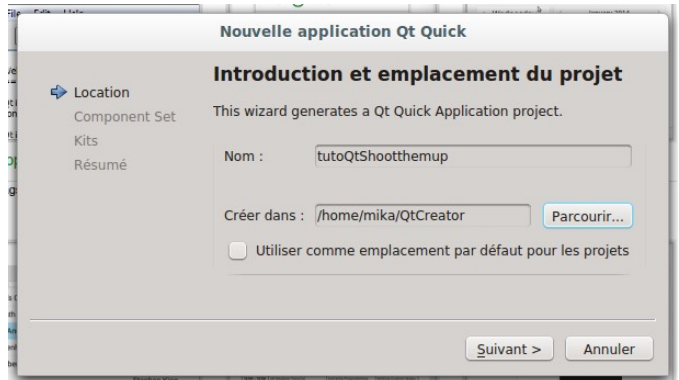
Une fois validé, vous en avez fini avec le SDK d'Android.

IV. Création d'un projet Qt Quick▲

Lancez Qt Creator.



Sélectionnez « créer un nouveau projet », sélectionnez dans le menu déroulant « modèle Android » et choisissez « Application Qt Quick ».



Ici, on sélectionne les kits de génération. Vous vous demandez peut-être pourquoi je n'ai pas sélectionné uniquement le kit Android. C'est simple : pour avoir déjà utilisé l'émulateur Android, je l'ai trouvé affreusement lent et donc contre-productif. Ici, l'idée est de profiter du côté multiplateforme de Qt pour générer tout au long du développement un binaire sur notre OS : rapide à générer et rapide à tester. Une fois le développement achevé, vous pouvez générer l'APK pour le tester/installer sur votre tablette.

Note : il peut être utile d'installer les bibliothèques de développement pour OpenGL. Pour cela, sous Ubuntu, vous pouvez utiliser la ligne suivante :

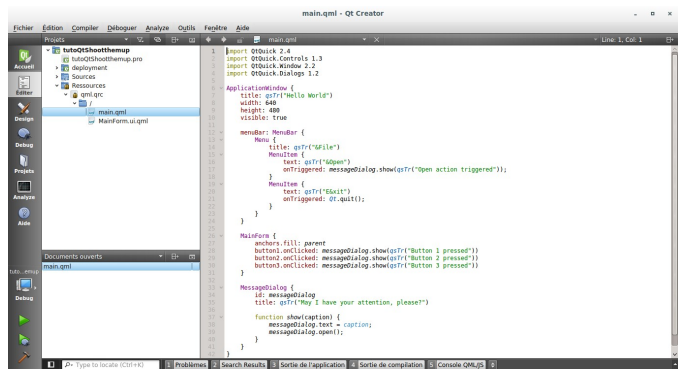
Sélectionnez

```
sudo apt-get install libgl1-mesa-dev
```

V. Découverte de Qt Creator, Qt Quick et JavaScript▲

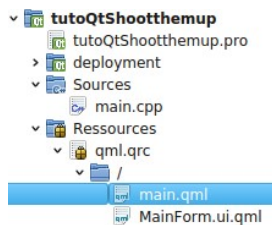
V-A. Interface de Qt Creator▲

Voici à quoi ressemble notre projet au démarrage :



- à gauche une liste de boutons/onglets permettant de passer d'un mode à l'autre : « éditer » pour voir le code source, « design » pour travailler sur l'interface... ;
- puis vient l'explorateur de fichiers du projet suivi de la visualisation du code ouvert ;
- enfin, en bas, les différentes sorties (compilation, exécution...).

En jetant un coup d'œil rapide sur l'arborescence, vous pouvez être étonné de ne pas retrouver de fichier JavaScript :



V-B. Ajout du fichier JavaScript▲

Au contraire, vous voyez bien des fichiers QML, C++, mais pas de fichier JavaScript.

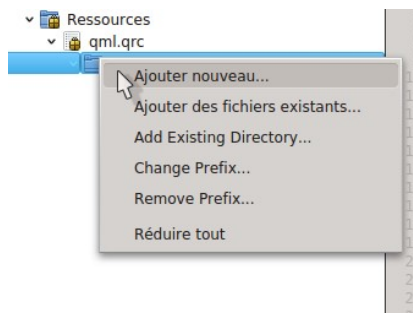
Avant de créer ce fichier, je vais ouvrir une parenthèse sur le JavaScript dans Qt Quick.

Votre application utilise C++ pour les traitements lourds, et des fichiers QML pour la couche présentation et ce sont ces fichiers QML qui permettent d'exécuter le JavaScript.

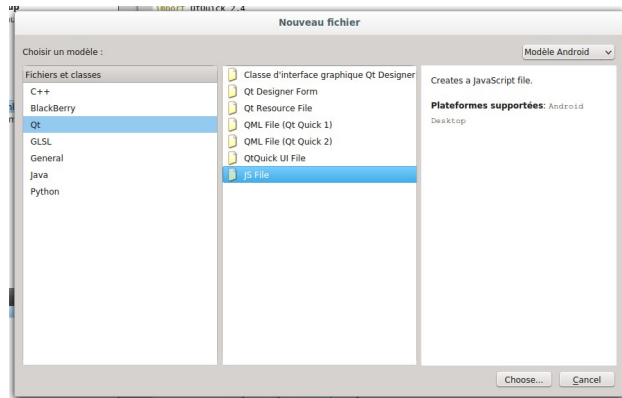
En effet, cela permet d'avoir à la fois un algorithme plus ou moins complexe et performant en C++ et une présentation / l'interface plus légère à développer, voire déléguée à un non-développeur (graphiste/designer).

Cela ressemble un peu à la manière de développer un site web : le contrôleur exécute la partie complexe/lourde (comme la connexion à la base de données), puis il affiche une page mêlant du HTML et du JavaScript. La différence, ici, est qu'on peut également interagir avec la base de données en Qt Quick.

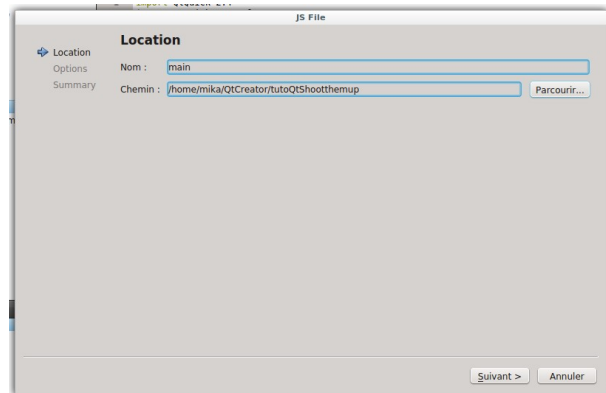
Ajoutez un fichier JavaScript à l'application : faites un clic droit sur le répertoire « / » dans « Ressources »



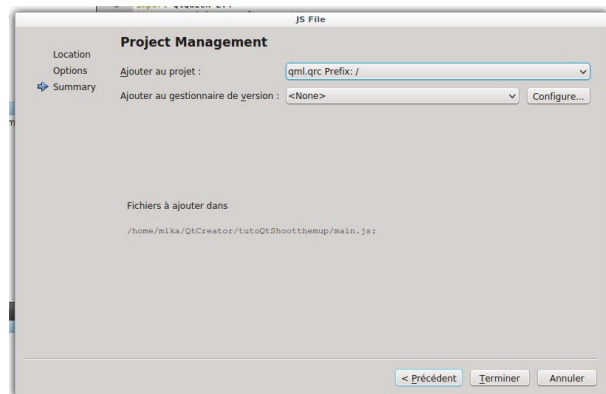
Vous avez ici une sélection de type de fichier : cliquez sur « Qt », puis sélectionnez « JS File »



Ensuite, indiquez le nom du fichier : « main ».

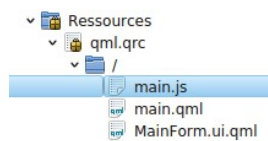


Validez plusieurs fois :



Vous avez la possibilité, comme vous pouvez le voir, de créer le fichier dans un autre répertoire, par exemple pour faire une arborescence plus structurée.

Ici, nous laisserons les choix par défaut.



Un nouveau fichier main.js fait donc logiquement son apparition dans l'arborescence précédente.

V-C. Inclure le fichier JavaScript dans le fichier QML▲

Éditez le fichier main.qml et ajoutez la ligne suivante :

Sélectionnez

```
import "Main.js" as Main
```

Dans notre fichier QML, cette ligne importe le fichier JavaScript dans l'espace de noms « Main » : vous pourrez ainsi appeler depuis l'interface vos fonctions/objets JavaScript.

V-D. Coder de la logique également au sein du QML avec les fonctions ▲

QML est bien plus qu'un simple fichier de mise en page, il permet de créer des composants, contenant d'autres objets et proposant des fonctions si nécessaire.

Ces fonctions, comme les autres éléments de QML, peuvent être contrôlées via notre code JavaScript.

On peut, par exemple, créer une fonction d'ouverture d'une boîte de dialogue en modifiant sa propriété de libellé puis en l'ouvrant ainsi (un exemple repris du fichier main.qml) :

Sélectionnez

```
MessageDialog {
    id: messageDialog
    title: qsTr("May I have your attention, please?")
    function showWindow(caption) {
        messageDialog.text = caption;
        messageDialog.open();
    }
}
```

On a ici un objet de type fenêtre de message. Il a un identifiant et une fonction personnelle qui permet changer son contenu et d'afficher la fenêtre.

Cette fonction peut être appelée dans le fichier QML, comme ici sur action des boutons :

Sélectionnez

```
MainForm {
    anchors.fill: parent
    button1.clicked: messageDialog.showWindow(qsTr("Button 1 pressed"))
    button2.clicked: messageDialog.showWindow(qsTr("Button 2 pressed"))
    button3.clicked: messageDialog.showWindow(qsTr("Button 3 pressed"))
}
```

On voit ici que l'on invoque la fonction implémentée dans l'élément précédent.

On peut modifier notre code présent pour voir la même interaction par du JavaScript, en modifiant le code JavaScript main.js comme suit :

Sélectionnez

```
function afficheDialogue(text_) {
    messageDialog.show(text_);
}
```

Et l'on modifie les appels des boutons ainsi :

Sélectionnez

```
MainForm {
    anchors.fill: parent
    button1.clicked: Main.afficheDialogue("Button 1 pressed")
    button2.clicked: Main.afficheDialogue("Button 2 pressed")
    button3.clicked: Main.afficheDialogue("Button 3 pressed")
}
```

On utilise ici l'objet « Main » précédemment initialisé par notre fichier JavaScript.

Comme vous pouvez le remarquer, on a bien accès aux éléments QML dans les deux sens, mais ce n'est pas tout : on peut également, côté JavaScript, créer de nouveaux objets QML.

Dans le jeu que nous allons voir, nous allons générer de nouveaux ennemis qui chuteront régulièrement et, pour cela, nous décrirons dans un premier temps ces ennemis dans un fichier QML, puis nous les afficherons à l'écran via le code JavaScript.

V-E. Invoquer un objet QML en JavaScript ▲

Nous allons créer un petit objet via un fichier QML, puis, côté JavaScript, nous en générerons deux instances à deux endroits différents

Ajoutez un fichier : Qt Quick 2.0, nommez le Enemy par exemple

Sélectionnez

```
import QtQuick 2.0
Rectangle {
    width: 100
    height: 62
    color: "#88118800"
}
```

Ajoutez maintenant dans notre fichier JavaScript deux fonctions pour invoquer des objets QML :

Sélectionnez

```
function createEnemy(x, y){
    var oComponent=Qt.createComponent('Enemy.qml');
    var oLogo=oComponent.createObject(root,{ "x": x_, "y": y_});
}
```

On utilise ici l'espace de nom Qt qui possède une méthode createComponent() prenant en paramètre le nom du fichier QML à instancier, puis nous utilisons sa méthode createObject() pour indiquer en premier paramètre où créer cette

instance, puis en second paramètre un objet pour forcer les propriétés de l'objet : ici ses coordonnées x,y.

Nous allons ajouter une seconde fonction pour invoquer deux fois cet objet à deux emplacements différents :

Sélectionnez

```
function createEnemies() {
    createEnemy(0,0);
    createEnemy(0,100);
}
```

Et nous allons appeler cette fonction avec l'un des boutons du fichier QML :

Sélectionnez

```
MainForm {
    anchors.fill: parent
    button1.onClicked: Main.createEnemies()
    button2.onClicked: Main.afficheDialog("Button 2 pressed")
    button3.onClicked: Main.afficheDialog("Button 3 pressed")
}
```

Une fois l'application générée, vous pouvez vérifier qu'en cliquant sur le premier bouton on crée bien deux instances issues de ce fichier QML. On peut ajouter du code sur le clic pour faire disparaître ces éléments, par exemple :

Sélectionnez

```
Rectangle {
    width: 100
    height: 62
    color: "#88118800"
    function kill() {
        console.log('kill');
        destroy();
    }
    MouseArea {
        anchors.fill: parent
        onClicked: kill()
    }
}
```

On ajoute ici deux choses pour que cela fonctionne : premièrement on définit une zone de clic où l'on précise que sur le clic on doit appeler une fonction kill().

Dans cette fonction kill(), on peut faire ce que l'on veut, ici on affiche dans la fenêtre de log « kill » puis on détruit l'objet.

En générant, vous allez pouvoir confirmer qu'en cliquant sur le premier bouton, deux rectangles sont créés, puis qu'en cliquant sur chacun des deux qu'ils sont bien supprimés.

VI. Conclusion ▲

Vous avez pu dans ce premier article découvrir un EDI et un écosystème à la fois puissant et facile à prendre en main.

Dans le second article, nous verrons concrètement le développement d'un jeu de type « shoot them up » avec cette technologie.

Vous voyez, ici, qu'il est assez facile, même aux jeunes développeurs, de développer leurs applications mobiles sans devoir apprendre pendant des semaines un nouveau langage (différent pour chaque plateforme).

VII. Remerciements ▲

Je souhaiterais remercier Thibaut Cuvelier et Alexandre Laurent pour leur soutien technique ainsi que lejimi et f-leb pour leur relecture orthographique.

Vous avez aimé ce tutoriel ? Alors partagez-le en cliquant sur les boutons suivants :



Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright © 2016 Michael Bertocchi. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

Responsable bénévole de la rubrique Qt : Thibaut Cuvelier - Contacter par email

Nous contacter Participez Hébergement Informations légales Partenaire : Hébergement Web
Copyright © 2000-2018 - www.developpez.com