1 Table des matières

2 REQUIREMENTS

2.1 (PC) Install Android Studio and NDK

https://developer.android.com/studio/index.html
https://developer.android.com/ndk/downloads/index.html

2.2 (PC) Install Visual Studio

2.3 (MAC) Install Xcode

https://itunes.apple.com/fr/app/xcode/id497799835?mt=12

2.4 (PC) Install console MSYS2

https://msys2.github.io/

Double click on the downloaded file and follow instructions.
Open MSYS2 shell: msys2_shell.cmd

```
pacman -Sy pacman
pacman -Syu
pacman -Su
pacman -S tar
pacman -S zip
pacman -S git
pacman -S base-devel
```

2.5 Install QT

https://www.qt.io/download-open-source/

Double click on the downloaded file and follow instructions.
(MAC) Move the created folder into your user account folder

2.6 Install QScintilla

https://www.riverbankcomputing.com/software/qscintilla/download

Double click on the downloaded file and follow instructions.
(PC) Unzip the file into the folder of your choice.
(MAC) Move the created folder into the Applications subfolder of your home directory.
(PC) Open MSYS2 shell
(MAC) Open a terminal window

(MAC) on the first install :

```
sudo xCodebuild -license
quit
agree
```

(MAC) WARNING ! if you use QScintilla 2.9.4 and QT 5.7 :

You need to change two files from QScintilla :
QScintilla_gpl-2.9.4/Qt4Qt5/qsciscintilla.cpp
QScintilla_gpl-2.9.4/Qt4Qt5/Qsci/qsciscintilla.h

See https://github.com/opencor/opencor/commit/70f3944e36b8b95b3ad92106aeae2f511b3f0e90

With the command line, go to Q4Qt5 subfolder of QScintilla :
Assuming X.x is your QT version, do the following :

On PC :

Y_y correspond à la version de mingw installée avec Qt (QTX.x/X.x/mingwY_y).
Open QT shell from here, (Y_y) being the MinGW version of your QT installation:
C:\Users\...\AppData\Roaming\Microsoft\Windows\StartMenu\Programs\Qt\X.x\MinGWY_y (32-bit)

(If needed, check the path in QTX.x/X.x/mingwY_y/bin/qtenv2.bat)

Navigate to Qt4Qt5 subfolder from QScintilla with 'cd' command, then type :

```
qmake qscintilla.pro
mingw32-make
mingw32-make install
```

On MAC :

```
~/Qt/X.x/clang_64/bin/qmake qscintilla.pro
make
make install
```

2.7 Install Git

http://git-scm.com/downloads

(PC) Double click on the downloaded file.
(MAC) Ctrl + right click, open with « Install program.app »
Follow instructions keeping default values as they are.
(PC) open git-bash.exe console
(MAC) open a terminal window

```
git config –global user.name "your git login"
git config –global user.email "votre git email"
```

2.8 Install Gideros

2.8.1 Fetch gideros source code

(PC) open git-bash console
(MAC) open a terminal

Navigate to the folder where you into to work.

```
git clone https://github.com/gideros/gideros
```

2.8.2 Configure scripts

Make a copy of the file gideros/scripts/Makefile.def.example and rename it into Makefile.def
Update the file copy according to your installation. Example:

On PC :

QTBASEDIR=/c/Applications/QT/QT58

QT_ARCH=mingw53_32
QT_TOOLSARCH=mingw530_32
QT5ICUVER=54 #Deprecated stuff
QTVER=5.8

For Android :
NDKBUILD=cmd //c /c/Applications/ANDROID_NDK/android-ndk-r10e/ndk-build.cmd
JAVA_HOME=C:/Program\ Files/Java/jdk1.8.0_92
ANT_HOME=C:/Applications/APACHE_ANT/apache-ant-1.9.7
ANDROID_HOME=C:/Applications/ANDROID_SDK/android-sdk

For WinRT :
FXC="/c/Program Files (x86)/Windows Kits/8.1/bin/x86/fxc.exe"
MSBUILD="/c/Program Files (x86)/MSBuild/14.0/Bin/MSbuild.exe"

For html5 :
EMSDK_PREFIX=cmd //c
CRUNCHME=crunchme-win32.exe

On MAC :

QTBASEDIR=~/Qt
QT_ARCH=clang_64
QT_TOOLSARCH=clang_64
QTVER=5.8
QT5ICUVER=54 #deprecated stuff

3 BUILD GIDEROS

3.1 On PC :

Open MSYS2 shell (msys2_shell.cmd)
Navigate to gideros folder

3.1.1 Build QT tools :

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
qtapp.install
```

Or if you have a suitable make program installed in MSYS2 :

```
make -f scripts/Makefile.gid qtapp.install
```

= GiderosStudio.exe, GiderosPlayer.exe, GiderosTexturePacker.exe (…) in Build.Win folder

3.1.2 Compilation Android :

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
android.install
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid android.install
```

= GiderosAndroidPlayer.apk dans dossier Build.Win/Players

3.1.3 Compilation winRT :

Si nécessaire :
Ouvrir la solution gideros/winRT_example/giderosgame.sln dans VisualStudio.
Vérifier la version de SQLite pour windows et windowsPhone et mettre à jour.
Enregistrer et fermer la solution.

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
winrt.install
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid winrt.install
```

= fichiers _bundle.appxupload dans dossier Build.Win/Players/WinRT

3.1.4 Compilation win32 :

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
win32.install
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid win32.install
```

= fichiers

3.1.5 Compilation html5 : à vérifier !

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
html5.install
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid html5.install
```

= fichiers

3.1.6 Compilation des plugins :

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
bundle.win
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid bundle.win
```

Erreurs temporaires en cours de correction :

```
un lien symbolique ne peut pas être créé vers « Versions/Current/GoogleMobileAds
un lien symbolique ne peut pas être créé vers « Versions/Current/Headers
```

= dossiers bin et fichiers gplugin dans dossier Build.Win/All Plugins

3.2 Sur MAC :

Ouvrez le Terminal.
cd vers le dossier « gideros »

3.2.1 Compilation QT :

```
make -f scripts/Makefile.gid qtapp.install
```

= GiderosStudio.exe, GiderosPlayer.exe, GiderosTexturePacker.exe (…) dans dossier Build.Mac

3.2.2 Compilation iOS :

```
make -f scripts/Makefile.gid ios.install
```

= GiderosiOSPlayer.zip dans dossier Buid.Mac/Players

3.2.3 Compilation des plugins :

```
make -f scripts/Makefile.gid bundle.mac
```

= dossiers bin et fichiers gplugin dans dossier Buid.Mac/All Plugins

A CHAQUE « PULL » DU GIT PENSEZ A RECOMPILER GIDEROS sur PC et sur MAC !

Reprenez la procédure à partir du point COMPILATION GIDEROS

4 CREATION PLUGIN

Vous pouvez utiliser les fichiers gideros/plugins/exampleplugin/ pour suivre cette partie du tuto.
Dans un objectif didactique ce plugin envoie deux types d'évènements :
_STATE lors de son démarrage et de son arrêt.
_WIFI après démarrage (nombre de wifi scannées, si le scan avait été autorisé ou non par la plateforme, la liste fixe des permissions nécessaires, la liste variable des permissions qui ont été vérifiées).

4.1 Fichier .gplugin

Créez un dossier avec le nom du plugin dans gideros/plugins/
Contenu du fichier gideros/plugins/exampleplugin/exampleplugin.gplugin :
Renseignez le nom du plugin et sa description.

```
<plugin
 name="Exampleplugin"
 description="description of the plugin">
</plugin>
```

Ce fichier sera à éditer pour chaque nouvelle plateforme à implémenter (Voir les points suivants).
Il contiendra alors les instructions requises lors de l'exportation du plugin par Gideros Studio.

4.2 Fichiers communs aux plateformes

Créez un sous-dossier gideros/plugins/exampleplugin/source/
Créez un sous-dossier ../source/common/

4.2.1 Le fichier binder.cpp :

Contenu du fichier ..source/common/examplepluginbinder.cpp :

Explications à partir du bas du fichier :
Les données d'**enregistrement** du plugin (« REGISTRER »).
La fonction d'**initialisation** du plugin (« g_initializePlugin »).
La fonction de **désinitialisation** du plugin (« g_deinitializePlugin »).

Explications à partir du haut du fichier :
Les références des **évènements*** possibles (« EXAMPLEPLUGIN_STATE », « EXAMPLEPLUGIN_WIFI »).
La **classe** du plugin et son héritage (« Exampleplugin : public GEventDispatcherProxy »).
Une fonction* (« start ») pour que le lua appelle la fonction de **démarrage** du fichier spécifique plateforme.
Une fonction* (« stop ») pour que le lua appelle la fonction d'**arrêt** du fichier spécifique plateforme.
Une fonction* (« test ») pour que le lua appelle la fonction de **test_binder** du fichier spécifique plateforme et pousse la donnée retournée en retour de l'appel de fonction lua (« test_lua »).

Cette fonction « test » va envoyer un booléen

Une fonction* (« gexampleplugin_dispatch ») qui envoie les données dans les évènements* lua.

La fonction de **chargement** du plugin (« loader ») qui :

- Liste les fonctions* et leurs références permettant de les appeler du lua (« ''test_lua'', test »).
- Appelle la fonction d'initialisation du fichier spécifique plateforme (« exampleplugin::init(»).
- Crée la classe (« Exampleplugin ») avec les références de fonctions* en paramètre.
- Pousse l'instance de la classe dans la pile lua.
- Pousse les évènements* possibles et leurs références dans la pile lua.

4.2.2 Le fichier binder.h :

Contenu du fichier ..source/common/examplepluginbinder.h :
Une **énumération** des évènements* possibles (« enum »).
Les **structures** des données transmises lors des évènements (« struct »).
La **fonction*** d'envoi des évènements (« gexampleplugin_dispatch »).
Le **namespace** du plugin (« exampleplugin »).
La liste des **fonctions** implémentées dans le fichier spécifique plateforme (« init », « deinit », « start », « stop », « test_binder »).

4.3 Fichiers spécifiques à chaque plateforme

4.3.1 Plateforme Android (Sur PC) :

Créez un sous-dossier gideros/plugins/exampleplugin/source/Android
Créez un sous-dossier ../Android/jni
Créez un sous-dossier ../Android/src/com/giderosmobile/android/plugins/exampleplugin/

Contenu du fichier ../Android/jni/gexampleplugin.cpp :

Explications à partir du bas du fichier :
Les **fonctions* définies et implémentées dans binder.h et binder.cpp** (« exampleplugin::test_binder() »).
Ces fonctions vont appeler les fonctions de la classe spécifique (« _gexampleplugin->test_plugin() »).
Les **fonctions natives Java** qui peuvent transmettre les données Java aux fonctions de la classe spécifique.

Explications à partir du haut du fichier :
La **classe spécifique** à la plateforme (« GExampleplugin »).
Le constructeur de la classe peut définir des **identifiants de méthodes Java** (« ArrayList_size_id »).
Les **signatures des méthodes Java** (''nom de la méthode'', '' (type des paramètres) type de retour'').
Les fonctions de la classe spécifique (« start() ») qui peuvent **appeler des méthodes Java** (« start_java() »)
Elles peuvent également **retourner des données** dans les fonctions* du binder.cpp (« test_plugin() ») .
Elles peuvent également utiliser les structures définies dans le binder.h pour **retourner des événements** (« onStateReceived() ») dans une fonction de retour d'événements (« callback ») qui va appeler la fonction* d'envoi des événements du binder.cpp (« gexampleplugin_dispatch(type, event) »).

Contenu du fichier ../Android/jni/Android.mk :
Référencement des **libs** gideros (« .so »).
Référencement du **fichier spécifique à la plateforme** (« gexampleplugin.cpp »).

Fichier ../Android/jni/Application.mk

Contenu du fichier ../Android/src/../plugins/exampleplugin/GExamplePlugin.java :
Référencement du **package**.
La **classe** Java (« GExamplePlugin »).
Les **méthodes Java nécessaires** (« onCreate », « onDestroy », « start_java », « stop_java », « test_java »).
Des **méthodes Java facultatives** (« onPause », onResume »).
Des **méthodes Java utiles** à appeler du fichier gexampleplugin.cpp (« getPermissionsChecked_java »).
Les **méthodes Java natives** (« onState ») à appeler du fichier .java et qui font retour dans les méthodes natives définies dans le fichier gexampleplugin.cpp.

Modifiez le fichier gideros/plugins/exampleplugin/exampleplugin.gplugin et y ajouter :

- Les instructions d'export pour Android Studio.
- Le référencement du plugin (« exampleplugin »).
- Le référencement du fichier .java (« GExampleplugin »).
- Le référencement des services nécessaires (« <service> »).
- Le référencement des receivers nécessaires (« <receiver> »).
- Le référencement des permissions nécessaires (« <uses-permission> »).

Passez au point COMPILATION Plugin Android

4.3.2 Plateforme winRT (Sur PC) :

Créez un sous-dossier gideros/plugins/exampleplugin/source/winRT

(En cours de rédaction)

Passez au point COMPILATION Plugin winRT

4.3.3 Plateforme iOS (Sur MAC) :

Créez un sous-dossier gideros/plugins/exampleplugin/source/iOS

(En cours de rédaction)

Passez au point COMPILATION Plugin iOS

5 COMPILATION PLUGIN

Après compilation, le plugin apparaitra dans la liste des plugins à exporter sous Gideros Studio.

X.x correspond à votre version de QT !

Sur PC : Ouvrez la console msys2_shell.cmd (cd vers le dossier « gideros »).
Z_z correspond à la version de mingw installée avec les Tools de QT (QTX.x/Tools/mingwZ_z)

Sur MAC : Ouvrez le Terminal (cd vers le dossier « gideros »)

5.1 Plugin Android (Sur PC)

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
nomDuPlugin.androidplugin
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid nomDuPlugin.androidplugin
```

Puis :

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
nomDuPlugin.androidplugin.install
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid nomDuPlugin.androidplugin.install
```

= dossiers bin et fichier gplugin dans dossier Buid.Win/All Plugins/nomDuPlugin/

Passez au point EXPORTATION Plugin Android

5.2 Plugin winRT (Sur PC)

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
nomDuPlugin.winrtplugin
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid nomDuPlugin.winrtplugin
```

Puis :

```
/c/QTX.x/Tools/mingwZ_z/bin/mingw32-make -f scripts/Makefile.gid
nomDuPlugin.winrtplugin.install
```

Ou si vous avez correctement installé pacman :

```
make -f scripts/Makefile.gid nomDuPlugin.winrtplugin.install
```

= dossiers bin et fichier gplugin dans dossier Buid.Win/All Plugins/nomDuPlugin/

Passez au point EXPORTATION Plugin winRT

5.3 Plugin iOS (Sur MAC)

```
make -f scripts/Makefile.gid nomDuPlugin.iosplugin
```

Puis :

```
make -f scripts/Makefile.gid nomDuPlugin.iosplugin.install
```

= dossiers bin et fichier gplugin dans dossier Buid.Mac/All Plugins/nomDuPlugin/

Passez au point EXPORTATION Plugin iOS

6 EXPORTATION PLUGIN

Vous pouvez utiliser le fichier gideros/plugins/exampleplugin/main.txt pour suivre cette partie du tuto.

Créez un nouveau projet Gideros Studio. Dans le fichier main.lua :

Implémentez l'initialisation et le chargement de votre plugin :

```
pcall(function () require("exampleplugin") end)
```

Implémentez les listeners et les fonctions dédiées, le démarrage et le test de votre plugin :

```
if exampleplugin then
	local function onEvent(event)
		--TODO
	end
	exampleplugin:addEventListener(Event.EXAMPLEPLUGIN_STATE, onEvent)
	exampleplugin:addEventListener(Event.EXAMPLEPLUGIN_WIFI, onEvent)
	exampleplugin.start_lua()
	print(exampleplugin.test_lua())
end
```

Ouvrez la fenêtre d'export du projet. Sélectionnez le plugin dans la liste des plugins à exporter.

A chaque modification du code lua, il est nécessaire de réexporter le plugin !

6.1 Plugin Android (Sur PC)

Choisissez l'architecture « Android » et le Template « Android Studio ».
Renseignez le nom de votre package.
Choisissez « Full Export » et cliquez sur OK. Exporter dans votre espace de travail Android Studio.

Sous Android Studio, il est possible de modifier et de tester le Manifest et le code Java sans avoir besoin de recompiler ni d'exporter (si ces modifications n'impliquent pas d'ajustements dans les fichiers autres que java !)
Après validation des tests, reportez vos modifications dans les fichiers du plugin sous gideros.
Puis recompilez et réexportez votre plugin en reprenant à partir du point COMPILATION Plugin Android.

6.2 Plugin winRT (Sur PC)

(En cours de rédaction)

Puis recompilez et réexportez votre plugin en reprenant à partir du point COMPILATION Plugin winRT.

6.3 Plugin iOS (Sur MAC)

(En cours de rédaction)

Puis recompilez et réexportez votre plugin en reprenant à partir du point COMPILATION Plugin iOS.