Table des matières

[2 PREREQUIS 3](#_Toc472516076)

[**2.1** (PC) Installation Android Studio et NDK 3](#_Toc472516077)

[**2.2** (PC) Installation Visual Studio 3](#_Toc472516078)

[**2.3** (MAC) Installation Xcode 3](#_Toc472516079)

[**2.4** (PC) Installation console MSYS2 3](#_Toc472516080)

[**2.5** Installation QT 3](#_Toc472516081)

[**2.6** Installation QScintilla 3](#_Toc472516082)

[**2.7** Installation Git 4](#_Toc472516083)

[**2.8** Installation Gideros 4](#_Toc472516084)

[2.8.1 Récupération du code source 4](#_Toc472516085)

[2.8.2 Récupération de code perso (facultatif) 4](#_Toc472516086)

[2.8.3 Paramétrage des scripts 5](#_Toc472516087)

[3 COMPILATION GIDEROS 6](#_Toc472516088)

[**3.1** Sur PC : 6](#_Toc472516089)

[3.1.1 Compilation QT : 6](#_Toc472516090)

[3.1.2 Compilation Android : 6](#_Toc472516091)

[3.1.3 Compilation winRT : 6](#_Toc472516092)

[3.1.4 Compilation win32 : 6](#_Toc472516093)

[3.1.5 Compilation html5 : à vérifier ! 7](#_Toc472516094)

[3.1.6 Compilation des plugins : 7](#_Toc472516095)

[**3.2** Sur MAC : 7](#_Toc472516096)

[3.2.1 Compilation QT : 7](#_Toc472516097)

[3.2.2 Compilation iOS : 7](#_Toc472516098)

[3.2.3 Compilation des plugins : 7](#_Toc472516099)

[4 CREATION PLUGIN 8](#_Toc472516100)

[**4.1** Fichier .gplugin 8](#_Toc472516101)

[**4.2** Fichiers communs aux plateformes 8](#_Toc472516102)

[4.2.1 Le fichier binder.cpp : 8](#_Toc472516103)

[4.2.2 Le fichier binder.h : 9](#_Toc472516104)

[**4.3** Fichiers spécifiques à chaque plateforme 9](#_Toc472516105)

[4.3.1 Plateforme Android (Sur PC) : 9](#_Toc472516106)

[4.3.2 Plateforme winRT (Sur PC) : 10](#_Toc472516107)

[4.3.3 Plateforme iOS (Sur MAC) : 10](#_Toc472516108)

[5 COMPILATION PLUGIN 11](#_Toc472516109)

[**5.1** Plugin Android (Sur PC) 11](#_Toc472516110)

[**5.2** Plugin winRT (Sur PC) 11](#_Toc472516111)

[**5.3** Plugin iOS (Sur MAC) 11](#_Toc472516112)

[6 EXPORTATION PLUGIN 12](#_Toc472516113)

[**6.1** Plugin Android (Sur PC) 12](#_Toc472516114)

[**6.2** Plugin winRT (Sur PC) 12](#_Toc472516115)

[**6.3** Plugin iOS (Sur MAC) 12](#_Toc472516116)

# PREREQUIS

## (PC) Installation Android Studio et NDK

<https://developer.android.com/studio/index.html>

<https://developer.android.com/ndk/downloads/index.html>

## (PC) Installation Visual Studio

## (MAC) Installation Xcode

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

## (PC) Installation console MSYS2

<https://msys2.github.io/>

Double clic sur fichier téléchargé et suivez les instructions.

Ouvrir console msys2\_shell.cmd

|  |
| --- |
| pacman –Sy pacman  pacman –Syu  pacman –Su  pacman –S tar wget  pacman –S zip unzip  pacman –S git  pacman –S base-devel |

## Installation QT

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

Double clic sur fichier téléchargé et suivez les instructions.

(MAC) Placer le dossier créé dans votre compte User/

## Installation QScintilla

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

Double clic sur fichier téléchargé et suivez les instructions.

(PC) Dézipper le fichier, extraire et déplacer dans le dossier souhaité.

(MAC) Placer le dossier créé dans votre compte User/Applications/

(PC) ouvrez console msys2\_shell.cmd

(MAC) ouvrez le Terminal

**(MAC) à la 1ère installation :**

|  |
| --- |
| sudo xCodebuild –license  quit  agree |

**(MAC) ATTENTION ! si vous utilisez QScintilla 2.9.4 et QT 5.7 :**

Il est nécessaire de modifier 2 fichiers dans QScintilla :

QScintilla\_gpl-2.9.4/Qt4Qt5/qsciscintilla.cpp

QScintilla\_gpl-2.9.4/Qt4Qt5/Qsci/qsciscintilla.h

Voir <https://github.com/opencor/opencor/commit/70f3944e36b8b95b3ad92106aeae2f511b3f0e90>

**En ligne de commande, se placer dans le sous dossier Q4Qt5 de QScintilla :**

X.x correspond à votre version de QT !

Sur PC :

Y\_y correspond à la version de mingw installée avec Qt (QTX.x/X.x/mingwY\_y).

Ouvrir la console QT à l’emplacement ci-dessous.

C:\Users\...\AppData\Roaming\Microsoft\Windows\StartMenu\Programs\Qt\X.x\MinGWY\_y (32-bit)

(Si nécessaire, voir le path dans le dossier QTX.x/X.x/mingwY\_y/bin/qtenv2.bat)

cd vers le dossier Qt4Qt5 de Scintilla

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

Sur MAC :

|  |
| --- |
| ~/Qt/X.x/clang\_64/bin/qmake qscintilla.pro  make  make install |

## Installation Git

<http://git-scm.com/downloads>

(PC) Double cliquer sur le fichier téléchargé.

(MAC) ctrl + click droit, ouvrir avec « Programme d’installation.app »

Suivez les instructions en laissant les valeurs par défaut.

(PC) ouvrez console git-bash.exe

(MAC) ouvrez le Terminal

|  |
| --- |
| git config –global user.name ‘’votre pseudo git‘’  git config –global user.email ‘’votre email git‘’ |

## Installation Gideros

### Récupération du code source

(PC) ouvrez console git-bash.exe

(MAC) ouvrez le Terminal

Se placer dans le dossier de travail souhaité.

|  |
| --- |
| git clone <https://github.com/gideros/gideros> |

### Récupération de code perso (facultatif)

(PC) ouvrez console git-bash.exe

(MAC) ouvrez le Terminal

Se placer dans le dossier gideros préalablement cloné.

(Ci-dessous, un exemple dans un dossier « gidperso » stocké sur le serveur « server.perso.com »)

|  |
| --- |
| GIT\_SSL\_NO\_VERIFY=true git submodule add <https://server.perso.com/git/gidperso> gidperso  git config http.sslVerify false (remettre valeur par défaut)  git status |

### Paramétrage des scripts

Faire une copie du fichier gideros/scripts/Makefile.def.example

Le renommer en « Makefile.def »

Modifier et adapter ce fichier (voir ci-dessous) pour pouvoir compiler Gideros.

**Sur PC :**

QTBASEDIR=/c/Applications/QT/QT58

QT\_ARCH=mingw53\_32

QT\_TOOLSARCH=mingw530\_32

QT5ICUVER=54

QTVER=5.8

Pour 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

Pour 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"

Pour html5 :

EMSDK\_PREFIX=cmd //c

CRUNCHME=crunchme-win32.exe

**Sur MAC :**

QTBASEDIR=~/Qt

MAC\_HOST=user@host

MAC\_PATH=gideros/gideros

QT\_ARCH=clang\_64

QT\_TOOLSARCH=clang\_64

QTVER=5.8

QT5ICUVER=54 =Variable temporaire (en cours de suppression)

# ****COMPILATION**** GIDEROS

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)

### Compilation QT :

|  |
| --- |
| /c/QTX.x/Tools/mingwZ\_z/bin/mingw32-make –f scripts/Makefile.gid qtapp.install |

Ou si vous avez correctement installé pacman :

|  |
| --- |
| make –f scripts/Makefile.gid qtapp.install |

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

### 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

### 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

### 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

### 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

### 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

## Sur MAC :

Ouvrez le Terminal.

cd vers le dossier « gideros »

### Compilation QT :

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

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

### Compilation iOS :

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

= GiderosiOSPlayer.zip dans dossier Buid.Mac/Players

### 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](#_COMPILATION_GIDEROS)

# 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).

## 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.

## Fichiers communs aux plateformes

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

Créez un sous-dossier ../source/common/

### 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.

### 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 »).

## Fichiers spécifiques à chaque plateforme

### 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](#_Toc472427701)

### Plateforme winRT (Sur PC) :

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

(En cours de rédaction)

Passez au point [COMPILATION Plugin winRT](#_Toc472427701)

### Plateforme iOS (Sur MAC) :

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

(En cours de rédaction)

Passez au point [COMPILATION Plugin iOS](#_Toc472427701)

# ****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 »)

## 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](#_EXPORTATION_PLUGIN)

## 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](#_EXPORTATION_PLUGIN)

## 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](#_EXPORTATION_PLUGIN)

# 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 !

## 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](#_Toc472427701).

## Plugin winRT (Sur PC)

(En cours de rédaction)

Puis recompilez et réexportez votre plugin en reprenant à partir du point [COMPILATION Plugin winRT](#_Toc472427701).

## Plugin iOS (Sur MAC)

(En cours de rédaction)

Puis recompilez et réexportez votre plugin en reprenant à partir du point [COMPILATION Plugin iOS](#_Toc472427701).