# Utilisation simple

Recopier l’intégralité de « Ovni\_Executable » dans un sous-répertoire de votre profil (accès en lecture et écriture).

Copier le sous-répertoire « BDDs » *de la fourniture de janvier 2020* également dans votre profil (non fourni sur github) : vous pouvez le placer dans le sous-répertoire d’Ovni ou à côté ou même sur un serveur externe. Note : une version plus complète existe déjà sur le serveur de Crira à l’Onera. Certains fichiers présents dans la version de janvier 2020 peuvent avoir été ajoutés ou modifiés depuis.

Lancer Ovni\_64.exe

Plus de détails dans Ovni\_Executable\Manuels\MU\_V5.docx (ou éventuellement dans les versions précédentes qui utilisaient Tcl/Tk).

# Installation pour compiler Ovni

## Fourniture de janvier 2020 :

On va ici recopier puis configurer un environnement de développement (CodeBlocks), un compilateur (gcc 8.1 en version MinGW 64 bits), la librairie wxWidgets 3.1.3 compilée mais aussi fournie avec les headers nécessaires à une compilation d’Ovni (ou tout autre programmes utilisant wxWidgets) et quelques outils Unix like. Cette étape ne nécessite pas forcément des droits d’administration, mais elle est manuelle. Elle n’est à faire qu’une seule fois … alors courage !

## Copies

Copier les sous-répertoires « MinGW64-8.1.0 », « Utilitaires\_Msys-Unix » et « wxWidgets-3.1.3 » à la racine de C: si possible (peut nécessiter des droits administrateur). Sinon, la racine de D: convient ou même un répertoire où vous avez accès en écriture. Attention à ce que le chemin d’accès ne comporte pas d’espace dans les noms intermédiaires et évitez aussi les caractères accentués : quelques utilitaires y sont réfractaires.

Copiez, si possible, « CodeBlocks\_wx313\_64 » dans « C:\Programmes » ou « C:\Program Files », là où résident les programmes 64 bits (si vous y avez accès en écriture). Sinon, sur D: ou tout autre sous-répertoire (de préférence sans espace ni caractère accentué).

Copier « Ovni\_Sources » dans votre profil, là où vous voulez, de préférences avec vos propres programmes.

Note : les noms des sous-répertoires peuvent être changés et simplifiés (par exemple MinGW64 ou MinGW seulement), mais il ne faudra pas oublier de les changer aussi dans toute la procédure ci-dessous.

## Fourniture via github

On va plutôt récupérer sur internet certains éléments.

## Compilateurs

**Version 8.1**

On peut récupérer MinGW64 en version 8.1 sur <https://sourceforge.net/projects/mingw-w64/files/>. La version utilisée dans Ovni est : x86\_64-posix-seh dont le chemin complet est :

<https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/8.1.0/threads-posix/seh/x86_64-8.1.0-release-posix-seh-rt_v6-rev0.7z>

Il suffit de dézipper et installer dans C:\MinGW64 par exemple.

**Version Msys2/9.3**

Msys2 est un environnement unix-like successeur de Msys mais un peu différent. On peut l’utiliser conjointement avec CodeBlocks ou en système autonome via une console. Il est fourni avec un utilitaire de mise à jour, pacman. Msys2 est maintenu régulièrement et suit les évolutions des compilateurs C, C++ ou Fortran. La version 8.1 du site MinGW64 sur SourceForge n’évolue plus depuis plus de 2 ans. C’est bien dommage et donc passer à Msys2 semble une bonne solution aujourd’hui.

Pour télécharger et installer Msys2, aller sur : <https://www.msys2.org/>.

Voir aussi : <https://solarianprogrammer.com/2019/11/05/install-gcc-windows/>

Par exemple, choisir/créer comme répertoire d’installation C:\msys64.

Dans ce répertoire, on trouve un exécutable msys2.exe qui ouvre une console. On y trouve aussi des répertoires mingw32, mingw64, clang32 … qui pour certains sont vides pour le moment.

L’installation des packages de base va se faire depuis <https://packages.msys2.org/base> ou pour plus de détails <https://packages.msys2.org/group/mingw-w64-x86_64-toolchain>

La commande pacman -S mingw-w64-x86\_64-toolchain doit en principe tout installer, mais on peut aussi installer les packages un par un.

La commande pacman –Syu permet les mises à jour.

Note : en fin de certaines mises à jour, la console semble bloquée. Il faut en sortir par la croix en haut à droite, comme indiqué dans la fenêtre.

L’exécutable mingw64.exe ouvre une fenêtre console dans laquelle on va vérifier les versions des compilateurs installés, en entrant par exemple :

gcc –v ou gcc –-version

g++ –v

gfortran –v

…

A noter : dans Msys2 on trouve des versions plus récentes de certains utilitaires qu’il y avait dans Utilitaires\_Msys-Unix .de la fourniture de janvier 2020. A voir, mais la double installation n’est pas forcément utile, du moins pour Ovni.

## wxWidgets

Pour le moment, c’est la version 3.1.3 qui est la version officielle. On peut la télécharger depuis le site :

<https://www.wxwidgets.org/downloads/>

à noter qu’une utilisation pour compiler CodeBlocks lui-même, nécessite une petite modification d’un fichier de configuration pour supporter une fonctionnalité dénommée Direct2D. C’est ce qui est fourni via github, ainsi que l’ensemble des include et librairies, utiles à la compilation d’Ovni. Le code source de wxWidgets n’est pas fourni sur github (voir la distribution officielle).

D’ici peu de temps, on devrait basculer en version 3.1.4. Les include et librairies compilées sont aussi fournies sur github, mais en version beta pour le moment.

## Variable PATH

L’accès à cette variable peut s’obtenir de plusieurs façons, toutes plus ou moins difficiles d’accès (Merci Microsoft !). Par exemple : depuis l’explorateur, clic droit sur « Ce PC » à gauche de l’écran, clic sur le sous-menu Propriétés, clic en haut à gauche sur « Page d’accueil du panneau de configuration » (seul élément sans droits administrateurs), clic sur « Comptes d’utilisateurs » et à gauche, clic sur « Modifier vos variables d’environnement ». On peut faire plus rapide si on sait accéder directement au « Panneau de configuration », par exemple via la fenêtre de recherche de « Panneau de configuration » en bas à gauche dans Windows 10.

Si vous êtes administrateurs, vous pouvez modifier le PATH système, sinon celui de l’utilisateur suffit. Dans ce cas, si la variable PATH n’existe pas, il suffit de la créer via le bouton « Nouvelle… ».

Y ajouter, plutôt au début, C:\MinGW64-8.1.0 ; C:\MinGW64-8.1.0\bin ; C:\Utilitaires\_Msys-Unix\bin

Notes :

* suivant la configuration de votre Windows, le séparateur de champs peut être ; ou , (mais sous Windows 10, ça se fait tout seul)
* le chemin C:\ est celui conseillé, mais si vous avez installé ces répertoires ailleurs, adaptez-les

## Configurer CodeBlocks

Lancer l’exécutable de CodeBlocks par un double clic sur codeblocks.exe, dans CodeBlocks\_wx313\_64 (ou là où vous l’avez copié).

Sur le site CodeBlocks, il existe aussi une version 20.03 récente qu’on peut télécharger et installer.

La première fois, CodeBlocks va tenter de trouver et configurer les compilateurs ainsi que proposer des associations avec les fichiers c, c++. Si vous n’avez pas d’autres compilateurs, vous pouvez choisir la dernière option, la totale ! Sinon laissez tel que, on y reviendra juste après.

Un message vous indiquera que SpellChecker doit être configuré. Pas la peine de s’en soucier pour le moment. Chez éventuellement la case en bas à gauche pour qu’on ne vous ennuie plus avec ça.

Settings / Environment / Global parameters (Paramètres / Environnement / Paramètres globaux) :

Cliquer sur le bouton Manage (Gérer …) sur la ligne « Check & Set file associations » (« Vérifier et choisir les associations…). On peut cocher l’ensemble des cases sauf celles correspondant à java ou MS Visual. Ainsi, les fichiers avec extension .c, .cpp, …, .cbp, …. seront associés à CodeBlocks. Moduler ces choix si vous avez Visual C et/ou Intel Fortran installés sur votre machine. Sortir par le bouton Appliquer. Vous pouvez aussi cliquer sur Set Now une fois revenu dans les general Settings.

Pour obtenir une interface en français, cliquer sur le menu Settings / Environment / View. Cocher « Internationalization » et choisir le français. Le changement ne sera effectif qu’au lancement suivant.

Settings / Compiler (Paramètres / Compilateur), aller dans l’onglet Toolchain executables (Programmes )

Si CodeBlocks a trouvé et configuré un compilateur GNU gcc, tant mieux. Néanmoins, faites en une copie que vous nommerez : GNU gcc compiler 64 (c’est ce qu’on utilisera par la suite) et que vous ajusterez ainsi :

Base  : C :\MinGW64-8.1.0[[1]](#footnote-1)

Compilateur C : gcc.exe

Compilateur C++  : g++.exe

Edition de liens lib dynamique : g++.exe

Edition de liens lib statique  : ar.exe

Débogueur  : Default\_64 (voir ci-dessous)

Compilateur de ressources  : windres.exe

Make  : mingw32-make.exe

Note : à la place de gcc et g++ on peut aussi mettre le nom complet comme x86\_64-w64-mingw32-gcc.exe et x86\_64-w64-mingw32-g++.exe (ou encore mingw32-gcc.exe et mingw32-g++.exe). Ce sont les mêmes exécutables.

Settings / debugger (Paramètres / Débogueur)

Créer Debug\_64 et y mettre le chemin de gdb, ici C:\MinGW64-8.1.0\bin\gdb.exe.

Cochez la case « évaluer l’expression sous le curseur » (cases cochées 2 à 5) et décochez éventuellement le premier choix (Disable startup script)

De retour dans Settings / Compiler (Paramètres / Compilateur), vous pouvez maintenant choisir Debug\_64 dans les Toolchains pour la ligne Debugger.

Settings / Environment / Environment variables (Paramètres / Environnement / Variables d’environnement) :

Créer, via le bouton Add en bas, une variable LC\_ALL avec la valeur en\_US.UTF-8. Cela évite une mauvaise interprétation par CodeBlocks de certains messages de gcc en cas de Warnings.

Settings/Global variables (Paramètres/Variables globales) :

Créer une variable wx3 puis mettre C:\ wxWidgets-3.1.3  dans la fenêtre base. Laisser le reste vide.

Settings/Compiler/Build Options (Paramètres/Compilateur/Options de génération) :

Augmenter le nombre de processus en parallèle : typiquement mettre 4 ou 8. Ce nombre doit, en théorie, être inférieur ou égal au nombre de processeurs (ou cœurs) de la machine. Permet à CodeBlocks de lancer en parallèle plusieurs compilations et donc de réduire le temps de compilation.

Fermez CodeBlocks et laissez-le sauvegarder les paramètres (layout). Ré-ouvrez CodeBlocks. Si vous avez opté pour l’interface en français, ça devrait être le cas maintenant.

Vous pouvez aussi vérifier que CodeBlocks a bien conservé les divers paramétrages vus ci-dessus.

Note : il faut peut-être réactiver LC\_ALL du fait du changement de langue.

Si vous allez dans Ovni\_Sources, vous devriez voir les icônes de CodeBlocks activées pour les fichiers .cbp, .cpp,…

Vous pouvez éliminer certains groupes d’icônes de la barre d’outils en haut, cela fera gagner un peu de place. Faire un clic droit dans un espace libre à côté de ces groupes d’icônes et décochez les cases NassiShneidermann, Code Completion (qui prend beaucoup de place) et éventuellement Fortran Projet. Vous pouvez alors réorganiser à la souris les différents groupes d’icônes.

Pour information, l’ensemble des paramètres de configuration de CodeBlocks est conservé dans un fichier de votre profil, quelque chose comme :

C:\Users\<nom utilisateur>\AppData\Roaming\CodeBlocks\Default.conf

## Test de compilation

Double cliquez sur le fichier Ovni\_wx3\_8.1.cbp.

CodeBlocks doit s’ouvrir sur le projet Ovni wxWidgets 64

Si vous avez bien exécuté les instructions de configuration, il ne devrait pas y avoir de soucis. Sinon, il faut changer le compilateur pour chacune des cibles. Pour cela, clic droit sur Ovni\_wx3 sur la gauche, sélectionner « Options de génération », puis pour chacune des cibles choisir le compilateur « GNU GCC compiler 64 ».

Note : la cible Release est celle à construire par défaut. Vous pouvez aussi choisir la cible Debug si vous voulez débuguer Ovni, ou Release Static, qui génèrera un exécutable statique, sans besoin de dlls annexes (plus pratique à transporter d’une machine à une autre mais beaucoup plus gros, ce qui ne convient pas toujours partout !).

Vous pouvez maintenant re-générer Ovni en cliquant sur l’icône en haut, avec 2 flèches en cercle, ou en passant par le menu Générer / Regénérer ou encore via la touche Ctrl-F11.

La nouvelle version a été créée dans le sous répertoire bin. Pour la lancer, cliquez sur la flèche verte en haut (Exécuter).

Si tout va bien, vous pouvez recopier le fichier Ovni\_64.exe de ce sous répertoire bin vers le répertoire Ovni\_executable. Normalement les dlls utiles y sont déjà.

# Association de fichiers pour Ovni

Il est pratique de lancer automatiquement Ovni en double cliquant sur un fichier de base de données d’avion. Les fichiers reconnus par Ovni en entrée sont :

.bdd : Format SDM Oktal, reconnu par Crira

.obj : Format Wavefront obj, très répandu

.3ds : Format Autodesk, binaire

.g3d : Format XML issu des travaux du Groupe 3D

.m3d : Milkshape 3D texte

.off : Object File Format (très vieux format, très limité.

.ply : Polygon File de Niratam

.dxf : Format texte très répandu. Toutefois, Ovni ne peut que les lire, pas les transformer. Support très partiel.

Rappel : Ovni ne crée en sortie que des fichiers de type .bdd.

Double cliquez sur un fichier .bdd. La première fois, il vous faudra indiquer le chemin vers Ovni\_64.exe. Par la suite, tous les fichiers .bdd seront ouverts par Ovni. Les fichiers sont reconnaissables par une petite icône d’avion, comme celle de l’exécutable Ovni.

Vous pouvez faire de même pour chacun des types reconnus. Faites attention avec les fichiers .obj. Cette extension est aussi utilisée par les compilateurs de Visual. Néanmoins, on ne « lance » pas un fichier Visual .obj par double clic. Il n’y a donc pas d’interaction fâcheuse, en principe !

Pour les associations suivantes, vous pouvez passer via un clic droit, par Ouvrir Avec… Dans la liste « Plus d’applications … » vous devriez trouver Ovni.

1. Si on a installé Msys2, la base sera plutôt C:\msys64\mingw64 [↑](#footnote-ref-1)